Lecture Notes in Computer Science 2572
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Barcelona*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

Diego Calvanese Maurizio Lenzerini
Rajeev Motwani (Eds.)

# Database Theory – ICDT 2003

9th International Conference
Siena, Italy, January 8-10, 2003
Proceedings

Springer

Volume Editors

Diego Calvanese
Maurizio Lenzerini
Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
E-mail: {calvanese/lenzerini}@dis.uniroma1.it

Rajeev Motwani
Department of Computer Science, Room 474
Gates Computer Science Building 4B
Stanford University, Stanford, CA 94305-9045, USA
E-mail: rajeev@cs.stanford.edu

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.2, F.1.3, F.4.1, I.2.1, H.4, F.2, H.3

ISSN 0302-9743
ISBN 3-540-00323-1 Springer-Verlag Berlin Heidelberg New York



Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de


Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna e. K.
Printed on acid-free paper SPIN: 10872297 06/3142 5 4 3 2 1 0

# Preface

This volume collects the papers presented at ICDT 2003, the 9th International Conference on Database Theory, held from 8 to 10 January 2003 in the Rettorato dell'Università di Siena, Siena, Italy.

ICDT (`http://alpha.luc.ac.be/~lucp1080/icdt/`) has now a long tradition of international conferences, providing a biennial scientific forum for the communication of high-quality research results on theoretical aspects of all forms of database systems and database technology. ICDT is traditionally held in historic European locations: Rome in 1986, Bruges in 1988, Paris in 1990, Berlin in 1992, Prague in 1995, Delphi in 1997, Jerusalem in 1999, and London in 2001. ICDT has merged with the Symposium on Mathematical Fundamentals of Database Systems (MFDBS), initiated in Dresden in 1987, and continued in Visegrad in 1989 and Rostock in 1991.

This volume contains 26 papers selected from 92 submissions, plus three invited papers by Hector Garcia-Molina, Yannis Ioannidis, and Limsoon Wong. We wish to thank all the authors who submitted papers, the members of the program committee for their efforts in reviewing and selecting the papers, the external referees, the organizing committee, and the sponsors for their support.

Diego Calvanese
Maurizio Lenzerini
Rajeev Motwani

# Organization

ICDT 2003 was organized by the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

## Organizing Committee

Maurizio Lenzerini (Chair)
Diego Calvanese
Marco Gori (Local Chair)
Tiziana Toni

## Program Co-chairs

Maurizio Lenzerini (University of Rome "La Sapienza", Italy)
Rajeev Motwani (Stanford University, U.S.A.)

## Program Committee

Serge Abiteboul (INRIA, France)
Foto Afrati (National Technical University of Athens, Greece)
Catriel Beeri (Hebrew University of Jerusalem, Israel)
Stephan Bressan (National University of Singapore, Singapore)
Peter Buneman (Univ. of Edinburgh, UK, and Univ. of Pennsylvania, USA)
Diego Calvanese (University of Rome "La Sapienza", Italy)
Surajit Chaudhuri (Microsoft Research)
Ronald Fagin (IBM Almaden Research Center, USA)
Alon Y. Halevy (University of Washinghton, USA)
Heikki Mannila (University of Helsinki, Finland)
Alberto O. Mendelzon (University of Toronto, Canada)
Guido Moerkotte (University of Mannheim, Germany)
Jeffrey F. Naughton (University of Wisconsin, USA)
Yannis Papakonstatinou (University of California, USA)
Luigi Palopoli (University of Reggio Calabria, Italy)
Domenico Saccà (University of Calabria, Italy)
Vladimir Sazonov (University of Liverpool, UK)
Bernhard Thalheim (Brandenburg Technical University at Cottbus, Germany)
Jeffrey D. Ullman (Stanford University, USA)
Jan Van den Bussche (Limburgs Universitair Centrum, Belgium)
Vasilis Vassalos (New York University, USA)

## External Referees

Fabrizio Angiulli
Marcelo Arenas
Brian Babcock
Stefano Basta
Michael Benedikt
Angela Bonifati
Francesco Buccafurri
Marco Cadoli
Andrea Calì
Paolo Ciaccia
Gregory Cobena
Gautam Das
Mayur Datar
Giuseppe De Giacomo
Wenfei Fan
Sergio Flesca
Paolo Franciosa
Irini Fundulaki
Filippo Furfaro
Venkatesh Ganti
Fosca Giannotti
Aris Gionis
Jonathan Goldstein
Dimitrios Gunopulos
Gianluigi Greco
Sergio Greco
Antonella Guzzo
Sven Helmer
Christoph Koch
Phokion Kolaitis
Nick Koudas
Gabriel Kuper
Giovambattista Ianni
Pasquale Legato
Domenico Lembo
Nicola Leone
Chen Li
Leonid Libkin
Alexei Lisitsa
Giuseppe Manco
Gerome Miklau
Tova Milo
Vivek Narasayya
Clara Pizzuti
Luigi Pontieri
Andrea Pugliese
Maurizio Rafanelli
Domenico Rosaci
Riccardo Rosati
Massimo Ruffolo
Pasquale Rullo
Alan P. Sexton
Francesco Scarcello
Luc Segoufin
Jerome Simeon
Domenico Talia
Domenico Ursino
Suresh Venkatasubramanian
Victor Vianu
Ester Zumpano

# Table of Contents

## Invited Papers



## Reasoning about XML Schemas and Queries



## Aggregate Queries



## Query Evaluation

## Query Rewriting and Reformulation



## Semistructured versus Structured Data



## Query Containment



## Consistency and Incompleteness

## Data Structures

# Open Problems in Data-Sharing Peer-to-Peer Systems

Neil Daswani, Hector Garcia-Molina, and Beverly Yang

Stanford University, Stanford CA 94305, USA,
{daswani, hector, byang}@db.stanford.edu,
http://www-db.stanford.edu

**Abstract.** In a Peer-To-Peer (P2P) system, autonomous computers pool their resources (e.g., files, storage, compute cycles) in order to inexpensively handle tasks that would normally require large costly servers. The scale of these systems, their "open nature," and the lack of centralized control pose difficult performance and security challenges. Much research has recently focused on tackling some of these challenges; in this paper, we propose future directions for research in P2P systems, and highlight problems that have not yet been studied in great depth. We focus on two particular aspects of P2P systems – *search* and *security* – and suggest several open and important research problems for the community to address.

## 1 Introduction

Peer-to-peer (P2P) systems have recently become a very active research area, due to the popularity and widespread use of P2P systems today, and their potential uses in future applications. Recently, P2P systems have emerged as a popular way to share huge amounts of data (e.g., [1,16,17]). In the future, the advent of large-scale ubiquitous computing makes P2P a natural model for interaction between devices (e.g., via the web services [18] framework).

P2P systems are popular because of the many benefits they offer: adaptation, self-organization, load-balancing, fault-tolerance, availability through massive replication, and the ability to pool together and harness large amounts of resources. For example, file-sharing P2P systems distribute the main cost of sharing data – bandwidth and storage – across all the peers in the network, thereby allowing them to scale without the need for powerful, expensive servers.

Despite their many strengths, however, P2P systems also present several challenges that are currently obstacles to their widespread acceptance and usage – e.g., security, efficiency, and performance guarantees like atomicity and transactional semantics. The P2P environment is particularly challenging to work in because of the scale of the network and unreliable nature of peers characterizing most P2P systems today. Many techniques previously developed for distributed systems of tens or hundreds of servers may no longer apply; new techniques are needed to meet these challenges in P2P systems.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 1–15, 2003.

In this paper, we consider research problems associated with *search* and *security* in *data-sharing* P2P systems. Though data-sharing P2P systems are capable of sharing enormous amounts of data (e.g., 0.36 petabytes on the Morpheus [17] network as of October 2001), such a collection is useless without a search mechanism allowing users to quickly locate a desired piece of data (Section 2). Furthermore, to ensure proper, continued operation of the system, security measures must be in place to protect against availability attacks, unauthentic data, and illegal access (Section 3). In this paper, we highlight several important and open research issues within both of these topics.

Note that this paper is not meant to be an exhaustive survey of P2P research. First, P2P can be applied to many domains outside of data-sharing; for example, computation (e.g., [19,20]), collaboration (e.g., [21]), and infrastructure systems (e.g., [22]) are all popular applications of P2P. Each application faces its own unique challenge (e.g., job scheduling in computation systems), as well as common issues (e.g., resource discovery). In addition, within data-sharing systems there exists important research outside of search and security. Good examples include resource management issues such as fairness and administrative ease. Finally, due to space limitations, the issues we present within search and security are not comprehensive, but illustrative. Examples are also chosen with a bias towards work done at the Stanford Peers group [23], because it is the research that the authors know best.

## 2 Search

A good *search mechanism* allows users to effectively locate desired data in a resource-efficient manner. Designing such a mechanism is difficult in P2P systems for several reasons: scale of the system, unreliability of individual peers, etc. In this section, we outline the basic architecture, requirements and goals of a search mechanism for P2P systems, and then suggest several areas of open research.

### 2.1 Overview

In a data-sharing P2P system, users submit queries and receive results (such as data, or pointers to data) in return, via the search mechanism. Data shared in the system can be of any type. In most cases users share files, such as music files, images, news articles, web pages, etc. Other possibilities include data stored in a relational DBMS, or a queryable spreadsheet. Queries may take any form that is appropriate given the type of data shared. For example, in a file-sharing system, queries might be keywords with regular expressions, and the search may be defined over different portions of the document (e.g., header, title, metadata).

A search mechanism defines the behavior of peers in three areas:

- **Topology:** Defines how peers are connected to each other. In some systems (e.g., Gnutella [1]), peers may connect to whomever they wish. In other systems, peers are organized into a rigid structure, in which the number and

nature of connections is dictated by the protocol. Defining a rigid topology may increase efficiency, but will restrict autonomy.
- **Data placement:** Defines how data or metadata is distributed across the network of peers. For example, in Gnutella, each node stores only its own collection of data. In Chord [2], data or metadata is carefully placed across nodes in a deterministic fashion. In super-peer networks [12], metadata for a small group of peers is centralized onto a single super-peer.
- **Message routing:** Defines how messages are propagated through the network. When a peer submits a query, the query message is sent to a number of the peer's "neighbors" (that is, nodes to whom the peer is connected), who may in turn forward the message sequentially or in parallel to some of their neighbors, and so on. When, and to whom, messages are sent is dictated by the routing protocol. Often, the routing protocol can take advantage of known patterns in topology and data placement, in order to reduce the number of messages sent.

In an actual system, the general model described above takes on a different form depending on the *requirements* of the system. Requirements are specified in several main categories:

- **Expressiveness:** The query language used for a system must be able to describe the desired data in sufficient detail. Key lookups are not expressive enough for IR searches over text documents, and keyword queries are not expressive enough to search structured data such as relational tables.
- **Comprehensiveness:** In some systems, returning any single result is sufficient (e.g., anycast), whereas in others, *all* results are required. The latter type of system requires a comprehensive search mechanism, in which all possible results are returned.
- **Autonomy:** Every search mechanism must define peer behavior with respect to topology, data placement, and message routing. However, autonomy of a peer is restricted when the mechanism limits behavior that a peer could reasonably expect to control. For example, a peer may wish to only connect to its friends or other trusted peers in the same organization, or the peer may wish to control which nodes can store its data (e.g., only nodes on the intranet), and how much of other nodes' data it must store. Depending on the purpose and users of the system, the search mechanism may be required to meet a certain level of autonomy for peers.

In this paper, we assume the additional requirement that the search mechanism be decentralized. A P2P system may have centralized search, and indeed, such "hybrid systems" have been very useful and popular in practice (e.g., [16]). However, centralized systems have been well-studied, and it is desirable that the search mechanism share the same benefits of P2P mentioned in Section 1; hence, here we focus only on decentralized P2P solutions.

While a well-designed search mechanism must satisfy the requirements specified by the system, it should also seek to maximize the following goals:

- **Efficiency:** We measure efficiency in terms of absolute resources consumed – bandwidth, processing power, storage, etc. An efficient use of resources results in lighter overhead on the system, and hence, higher throughput.
- **Quality of Service:** We can measure quality of service (QoS) along many different metrics depending on the application – number of results, response time, etc. Note the distinction between QoS and efficiency: QoS focuses on user-perceived qualities, while efficiency focuses on the resource cost (e.g., bandwidth) to achieve a particular level of service.
- **Robustness:** We define robustness to mean stability in the presence of failures: quality of service and efficiency are maintained as peers in the system fail or leave. Robustness to attacks is a separate issue discussed in Section 3.

By placing current work in the framework of requirements and goals above, we can identify several areas in which research is much needed. In the following section, we mention just a few of these areas.

### 2.2 Expressiveness

In order for P2P systems to be useful in a wide range of applications, they must be able to support query languages of varying levels of expressiveness. Thus far, work in P2P search has focused on answering simple queries, such as key lookups. An important area of research therefore lies in developing mechanisms for richer query languages. Here, we list a few examples of useful types of queries, and discuss the related work and challenges in supporting them.

- **Key lookup:** The simplest form of query is an object lookup by key or identifier. Protocols directly supporting this primitive have been widely studied, and efficient solutions exist (e.g., [2,3,4]). Ongoing research is exploring how to make these protocols more efficient and robust [5].
- **Keyword:** While much research has focused on search techniques for keyword queries (e.g., [11,10,6]), all of these techniques have been geared towards efficient, *partial* (not comprehensive) search – e.g., all music-sharing systems currently only support partial search. Partial search is acceptable in those applications where a few keywords can usually uniquely identify the desired file (e.g., music-sharing systems, as opposed to web page repositories), because the first few matches are likely to satisfy the user's request. Techniques for partial search can always be made comprehensive simply by sending the query message to every peer in the network; however, such an approach is prohibitively expensive. Hence, designing techniques for efficient, comprehensive search remains an open problem.
- **Ranked keyword:** If many results are returned for comprehensive keyword search, users will need results to be ranked and filtered by relevance. While the query language for ranked keyword search remains the same, the additional information in the results (i.e., the relevance ranking) poses additional challenges and opportunities. For example, ranked search can be built on top of regular search by retrieving all results and sorting locally; however, state-of-the-art ranking functions usually require global statistics over the total

collection of documents (e.g., document frequency). Collecting and maintaining these statistics in a robust, efficient, and distributed manner is a challenge. At the same time, ranked results allow the system to return "top k" results, which provides the opportunity to optimize search if $k$ is much less than the total number of results (which is generally the case, for example, in web searches). Techniques for ranked search exists for distributed systems of moderate scale (e.g., [7]), but future research must extend these techniques to support much larger systems.

- **Aggregates:** A user may sometimes be interested in knowing *aggregate* properties of the system or data collection as a whole, rather than locating specific data. For example, to collect global statistics to support ranked keyword search mentioned earlier, a user could submit several `SUM` queries to sum the number of documents that contain a particular term. Ongoing research [8] addresses `COUNT` queries defined over a predicate – for example, counting the number of nodes that belong to the `stanford.edu` domain. Further research is needed to extend these techniques into more expressive aggregates like `SUM`, `MAX`, and `MEDIAN`.
- **SQL:** As a complex language defined over a rich data model, SQL is the most difficult query language to support among the examples listed. Current research on supporting SQL in P2P systems is very preliminary. For example, the PIER project [9] supports a subset of SQL over a P2P framework, but they report significant performance "hotspots" in their preliminary implementation. A great deal of additional research is needed to advance current work into a search mechanism with reasonable performance, and to investigate alternative approaches to the problem.

### 2.3 Autonomy, Efficiency, and Robustness

Autonomy, efficiency and robustness are all desirable features in any system. These features conceptually define an informal space of P2P systems, as shown in Figure 1*a*, where a point in the space represents a system with the corresponding "values" for each feature. Note that the value of a system with respect to a feature only provides a partial order, since features can be measured along several metrics (e.g., efficiency can be measured by bandwidth, processing power, and storage). Hence, Figure 1 illustrates the space by showing just a few points for which the relative order (and not the actual coordinates) along each feature is fairly obvious.

The space defined by autonomy, efficiency and robustness is not fully explored; in particular, there appears to be some correlation between autonomy and efficiency (Figure 1*b*), and autonomy and robustness (Figure 1*c*). A partial explanation for the first correlation is that less autonomy allows the search mechanism to specify a data placement and topology such that:

- There exist a deterministic way to locate data within bounded cost (e.g., Chord)
- There is a small set of nodes that is guaranteed to hold the answer, if it exists (e.g., super-peer networks, concept clusters [13])

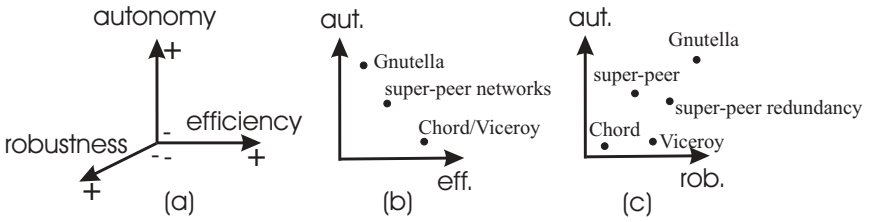



**Fig. 1.** The space of systems defined by autonomy, efficiency and robustness (*a*). Looking at a few example systems within this space, there appears to be a relationship between autonomy and efficiency *(b)*, and autonomy and robustness (*c*)

– There is an increased chance of finding results on a random node (e.g., replication [6]).

At the same time, these rigidly organized networks can be difficult or expensive to maintain, especially as peers join and leave the network at the rapid rate characteristic of many P2P systems. As a result, robustness is also correlated with autonomy.

One important area of research is finding techniques that push beyond the current tradeoffs between efficiency, autonomy and robustness. Decoupling efficiency from autonomy seems to be the greatest challenge, since existing techniques almost uniformly sacrifice autonomy to achieve efficiency. However, the potential gain is the greatest: a search mechanism that is efficient, robust, *and* preserves peer autonomy. Decoupling autonomy from robustness is also important, because it allows greater flexibility in choosing the desired properties of the mechanism. For example, a search mechanism that is robust, but has low peer autonomy, can be desirable if the lack of autonomy leads to efficiency, and peer autonomy is not a requirement of the system.

Several research projects have tackled the autonomy/robustness tradeoff. For example, the Viceroy [14] network construction maintains a low level of peer autonomy, but increases robustness and efficiency by reducing the cost of maintaining the network structure to a constant term, for each join/leave of a peer. In comparison, most distributed hash tables (DHTs) with the same functionality have logarithmic maintenance cost. As another example, super-peer redundancy [12] imposes slightly stricter rules on topology and data placement within a cluster of peers, but this decrease in autonomy results in greater robustness of the super-peer and improved efficiency in the overall network.

Another interesting area of research is providing fine-granularity tuning of the tradeoff between autonomy and efficiency within a single system. A single user may have varying needs; for example, a company may have a few sensitive files that must remain on the intranet, but the remaining files can be stored anywhere. A single system that can be tuned to support all of these needs is more desirable than requiring users to use different systems for different purposes. A good example of a tunable system is SkipNet [15]. SkipNet allows users to specify a range of peers on which a document may be stored (e.g., all peers within the

`stanford.edu` domain). At one extreme, if the range is always limited to a single peer, then user autonomy is high, but the system ceases to be P2P and loses good properties such as load-balancing and self-organization. At the other extreme, if the range always includes all peers in the network, SkipNet functions as a traditional P2P lookup system with low autonomy, but other good properties. While SkipNet does not push beyond existing tradeoffs, its value lies in allowing users to choose the point along the tradeoff that meets their needs.

### 2.4 Quality of Service

In the previous discussion, we implicitly assume a fixed level of service (e.g., number of results per query) that must be maintained as other factors (e.g., autonomy) are varied. However, quality of service (QoS) can be measured with many different metrics, depending on the application, and a spectrum of acceptable performance exists along each metric. Examples of service metrics include number of results (e.g., in partial-search systems), response time, and relevance (e.g., precision and recall in ranked keyword searches). A constant challenge in designing P2P systems is achieving a desired level of QoS as efficiently as possible. Because metrics and applications differ so widely, this challenge must often be tackled on a per-case basis.

As an example, the number of results returned is an important QoS metric for partial-search systems like Gnutella. However, in systems where there is high autonomy (such as Gnutella), there is a clear and unavoidable tradeoff between number of results and cost; hence, the interesting problem is to get as close as possible to the lower bounds of the tradeoff. For example, the directed BFS technique in [11] attempts to minimize cost by sending messages to "productive" nodes (e.g., nodes with large collections). Concept-clustering networks (e.g., [13]) cluster peers together according to "interest" (e.g., music genre), and send queries to the cluster that best matches the queries' area of interest. These techniques do improve the tradeoff between cost and number of results, but are clearly not optimal: performance of directed BFS depends on the ad-hoc topology and is therefore unpredictable, while concept-clustering only works well if queries and interests fall cleanly into single categories. Can there exist a general technique that can guarantee (with high probability) that the cost/QoS tradeoff is optimal?

With other metrics of QoS, there is not such an obvious tradeoff between quality and cost. In these cases, the goal is to maintain the same level of service while decreasing cost. For example, consider the "satisfaction" metric, which is binary and is true when a threshold number of results is found. Satisfaction is an important metric in partial-search systems where only the first $k$ results are displayed to the user (e.g., [16,1]). Reference [11] shows that, compared to current techniques, cost can be drastically reduced while maintaining satisfaction. Furthermore, even better performance is probably possible if we discard this work's requirement of peer autonomy and simplicity. Additional research is required to explore this space further.

# 3 Security

Securing P2P data sharing applications is challenging due to their open and autonomous nature. Compared to a client-server system in which servers can be relied upon or trusted to always follow protocols, peers in a P2P system may provide no such guarantee. The environment in which a peer must function is a hostile one in which any peer is welcome to join the network; these peers cannot necessarily be trusted to route queries or responses correctly, store documents when asked to, or serve documents when requested. In this part of the paper, we outline a number of security issues that are characteristic to P2P data sharing systems, discuss a few examples of research that has taken place to address some of these issues, and suggest a number of open research problems.

We organize the security requirements of P2P data sharing systems into four general areas: availability, file authenticity, anonymity, and access control. Today's P2P systems rarely address all of the necessary requirements in any one of these areas, and developing systems that have the flexibility to support requirements in all of these areas is expected to be a research challenge for quite some time.

For each of these areas, it will be important to develop techniques that *prevent*, *detect*, *manage*, and are able to *recover* from attacks. For example, since it may be difficult to prevent a denial-of-service attack against a system's availability, it will be important to develop techniques that are able to 1) detect when a denial-of service attack is taking place (as opposed to there just being a high load), 2) manage an attack that is "in-progress" such that the system can continue to provide some (possibly reduced) level of service to clients, and 3) recover from the attack by disconnecting the malicious nodes.

## 3.1 Availability

There are a number of different node and resource availability requirements that are important to P2P file sharing systems. In particular, each node in a P2P system should be able to accept messages from other nodes, and communicate with them to offer access to the resources that it contributes to the network.

A denial-of-service (DoS) attack attempts to make a node and its resources unavailable by overloading it. The most obvious DoS attack is targeted at using up all of a node's bandwidth. This type of attack is similar to traditional network-layer DoS attacks (e.g. [31]). If a node's available bandwidth is used up transferring useless messages that are directly or indirectly created by a malicious node, all of the other resources that the node has to offer (including CPU and storage) will also be unavailable to the P2P network.

A specific example of a DoS attack against node availability is a chosen-victim attack in Gnutella that an adversary constructs as follows: a malicious super-node maneuvers its way into a "central" position in the network and then responds to every query that passes thru it claiming that the victim node has a file that satisfies the query (even though it does not). Every node that receives one of these responses then attempts to connect to the victim to obtain the

file that they were looking for, and the number of these requests overloads the bandwidth of the victim such that any other node seeking a file that the victim does have is unable to communicate the victim.

The key aspect to note here is that in our example the attacker exploited a vulnerability of the Gnutella P2P protocol (namely, that any node can respond to any query claiming that any file could be anywhere). In the future, P2P protocols need to be designed to make it hard for adversaries to construct DoS attacks by taking advantage of loosely constrained protocol features.

Attackers that construct DoS attacks typically need to find and take advantage of an "amplification mechanism" in the network to cause significantly more damage than they could with only their own resources. In addition, if they would like to have control over how their attack is carried out, they must also find or create a back-door communication channel to communicate with "zombie" hosts that they infiltrate using manual or automatic means. It is important to design future P2P protocols such that they do not open up new opportunities for attackers to use as amplifiers and back-door communication channels.

Some research has taken place to date to specifically address DoS attacks in P2P networks. In particular, [38] addresses DoS attacks based on query-floods in the Gnutella network. However, more research is necessary to understand the effects of other types of DoS attacks in various P2P networks.

Aside from DoS attacks, node availability can also be attacked by malicious users that infiltrate victim nodes and induce their failure. These types of attacks can be modeled as fail-stop or byzantine failures, which could potentially be dealt with using many techniques that have already been developed (e.g. [34]). However, these techniques have typically not been popular due to their inefficiency, unusually high message overhead, and complexity. In addition, these techniques often assume complete and secure pairwise connectivity between nodes, which is not the case in most P2P networks. Further research will be necessary to make these or similar techniques acceptable from a performance and security standpoint in a P2P context.

In addition, there are many proposals to provide significant levels of fault-tolerance in the face of node failure including CAN [3], Chord [2], Pastry [4], and Viceroy [14]. Security analyses of these types of proposals can be found in [43] and [36]. The IRIS [25] project seeks to continue the investigation of these types of approaches.

A malicious node can also directly attack the availability of any of the particular resources at a node. The CPU availability at a node can be attacked by sending a modest number of complex queries to bog down the CPU of a node without consuming all of its bandwidth. The available storage could be attacked by malicious nodes who are allowed to submit bogus documents for storage. One approach to deal with this is to allocate storage to nodes in a manner proportional to the resources that a node contributes to the network as proposed in [28].

We might like to ensure that all files stored in the system are always available regardless of which nodes in the network are currently online. File availability

ensures that files can be perpetually preserved, regardless of factors such as the popularity of the files. Systems such as Gnutella and Freenet provide no guarantees about the preservation of files, and unpopular files tend to disappear.

Even if files can be assured to physically exist and are accessible, a DoS attack can still be made against the quality-of-service with which they are available. In this type of a DoS attack, a malicious node makes a file available, but when a request to download the file is received, it serves the file so slowly that the requester will most likely lose patience and cancel the download before it completes. The malicious node could also claim that it is serving the file requested but send some other file instead. As such, techniques such as hash trees [26] could to be used by the client to incrementally ensure that the server is sending the correct data, and that data is sent at a reasonable rate.

### 3.2 File Authenticity

File authenticity is a second key security requirement that remains largely unaddressed in P2P systems. The question that a file authenticity mechanism answers is: given a query and a set of documents that are responses to the query, which of the responses are "authentic" responses to the query? For example, if a peer issues a search for "Origin of Species" and receives three responses to the query, which of these responses are "authentic"? One of the responses may be the exact contents of the book authored by Charles Darwin. Another response may be the content of the book by Charles Darwin with several key passages altered. A third response might be a different document that advocates creationism as the theory by which species originated.

Note that the problem of file authenticity is different than the problem of file (or data) integrity. The goal of file integrity is to ensure that documents do not get inadvertently corrupted due to communication failures. Solutions to the file integrity problem usually involve adding some type of redundancy to messages in the form of a "signature." After a file is sent from node A to node B, a signature of the file is also sent. There are many fewer bits in the signature than in the file itself, and every bit of the signature is dependent on every bit of the file. If the file arrived at node B corrupted, the signature would not match. Techniques such as CRCs (cyclic redundancy checks), hashing, MACs (message authentication codes), or digital signatures (using symmetric or asymmetric encryption) are well-understood solutions to the file integrity problem.

The problem of file authenticity, however, can be viewed as: given a query, what is (or are) the "authentic" signature(s) for the document(s) that satisfy the query? Once some file authenticity algorithm is used to determine what is (or are) the authentic signatures, a peer can inspect responses to the query by checking that each response has an authentic signature.

In our discussion until this point, we have not defined what it means for a file to be authentic. There are a number of potential options: we will outline four reasonable ones.

*Oldest Document.* The first definition of authenticity considers the oldest document that was submitted with a particular set of metadata to be the authentic

copy of that document. For example, if Charles Darwin was the first author to ever submit a document with the title "Origin of Species," then his document would be considered to be an authentic match for a query looking for "Origin of Species" as the title. Any documents that were submitted with the title "Origin of Species" after Charles Darwin's submission would be considered unauthentic matches to the query even if we decided to store these documents in the system. Timestamping systems (e.g. [35]) can be helpful in constructing file authenticity systems based on this approach.

*Expert-Based.* In this approach, a document would be deemed authentic by an "expert" or authoritative node. For example, node G may be an expert that keeps track of signatures for all files ever authored by any user of G. If a user searching for documents authored by any of G's users is ever concerned about the potential authenticity of a file received as a response to a query, node G can be consulted. Of course, if node G is unavailable at any particular time due to a transient or permanent failure, is infiltrated by an attacker, or is itself malicious, it may be difficult to properly verify the authenticity of files that G's users authored. Offline digital signature schemes (i.e., RSA) can be used to verify file authenticity in the face of node failures, but are limited by the lifetime and security of public/private keys.

*Voting-Based.* To deal with the possible failure of G or a compromised key in our last approach, our third definition of authenticity takes into account the "votes" of many experts. The expert nodes may be nodes that are run by human experts qualified to study and assess the authenticity of particular types of files, and the majority opinion of the human experts can be used to assess the authenticity of a file. Alternatively, the expert nodes may simply be "regular" nodes that store files, and will vote that a particular file is authentic if they store a copy of it. In this scheme, users are expected to delete files that they do not believe are authentic, and a file's authenticity is determined by the number of copies of the file that are distributed throughout the system. The key technical issues with this approach are how to prevent spoofing of votes, of nodes, and of files.

*Reputation-Based.* Some experts might be more trustworthy than others (as determined by past performance), and we might weight the votes of more trustworthy experts more heavily. The weights in this approach are a simple example of "reputations" that may be maintained by a reputation system. A reputation system is responsible for maintaining, updating, and propagating such weights and other reputation information [41]. Reputation systems may or may not choose to use voting in making their assessments. There has been some study of reputation systems in the context of P2P networks, but no such system has been commercially successful (e.g. [33,24]).

### 3.3 Anonymity

There is much work that has taken place on anonymity in the context of the Internet both at the network-layer (e.g. [30]) as well as at the application-layer

**Table 1.** Types of Anonymity

| *Type of Anonymity* | *Difficult for Adversary to Determine:* |
|---|---|
| Author | Which users created which documents? |
| Server | Which nodes store a given document? |
| Reader | Which users access which documents? |
| Document | Which documents are stored at a given node? |

(e.g. [40]). In this section we specifically focus on application-layer anonymity in P2P data sharing systems.

While some would suggest that many users are interested in anonymity because it allows them to illegally trade copyrighted data files in an untraceable fashion, there are many legitimate reasons for supporting anonymity in a P2P system. Anonymity can enable censorship resistance, freedom of speech without the fear of persecution, and privacy protection. Malicious parties can be prevented from deterring the creation, publication, and distribution of documents. For example, such a system may allow an Iraqi nuclear scientist to publish a document about the true state of Iraq's nuclear weapons program to the world without the fear that Saddam Hussein's regime could trace the document back to him or her. Users that access documents could also have their privacy protected in such a system. An FBI agent could access a company's public information resources (i.e., web pages, databases, etc.) anonymously so as not to arouse suspicion that the company may be under investigation.

There are a number of different types of anonymity that can be provided in a P2P system. It is difficult for the adversary to determine the answers to different questions for different types of anonymity. Table 1 summarizes a few types of anonymity discussed in [39].

We would ideally like to provide anonymity while maintaining other desirable search and security features such as efficiency, decentralization, and peer discovery. Unfortunately, providing various types of anonymity often conflicts with these design goals for a P2P system.

To illustrate one of these conflicting goals, consider the natural trade-off between server anonymity and efficient search. If we are to provide server anonymity, it should be impossible to determine which nodes are responsible for storing a document. On the other hand, if we would like to be able to efficiently search for a document, we should be able to tell exactly which nodes are responsible for storing a document. A P2P system such as Free Haven that strives to provide server anonymity resorts to broadcast search, while others such as Freenet [27] provide for efficient search but do not provide for server anonymity. Freenet does, however, provide author anonymity. Nevertheless, supporting server anonymity and efficient search concurrently remains an open issue.

There exists a middle-ground: we might be able to provide some level of server anonymity by assigning pseudonyms to each server, albeit at the cost of search efficiency. If an adversary is able to determine the pseudonym for the server of

a controversial document, the adversary is still unable to map the pseudonym to the publisher's true identity or location. The document can be accessed in such a way as to preserve the server's anonymity by requiring that a reader (a potential adversary) never directly communicate with a server. Instead, readers only communicate with a server through a chain of intermediate proxy nodes that forward requests from the reader to the server. The reader presents the server's pseudonym to a proxy to request communication with the server (thereby hiding a server's true identity), and never obtains a connection to the actual server for a document (thereby hiding the server's location). Reader anonymity can also be provided using a chain of intermediate proxies, as the server does not know who the actual requester of a document is, and each proxy does not know if the previous node in the chain is the actual reader or is just another proxy. Of course, in both these cases, the anonymity is provided based on the assumption that proxies do not collude. The degradation of anonymity protocols under attacks has been studied in [44], and this study suggests that further work is necessary in this area.

Free Haven and Crowds [40] are examples of systems that use forwarding proxies to provide various types of anonymity with varying strength. Each of these systems differ in how the level of anonymity degrades as more and more potentially colluding malicious nodes take on the responsibilities of proxies. Other techniques that are commonly found in systems that provide anonymity include mix networks (e.g. [32]), and using cryptographic secret-sharing techniques to split files into many shares (e.g. [42]).

### 3.4 Access Control

Intellectual property management and digital rights management issues can be cast as access control problems. We want to restrict the accessibility of documents to only those users that have paid for that access. P2P systems currently cannot be trusted to successfully enforce copyright laws or carry out any form of such digital rights management, especially since few assumptions can be made about key management infrastructure. This has led to blatant violation of copyright laws by users of P2P systems, and has also led to lawsuits against companies that build P2P systems.

The trade-offs involved in enforcing access control in a P2P data sharing system are challenging because if a system imposes restrictions over what types of data it shares (i.e., only copy-protected content), then its utility will be limited. On the other hand, if it imposes no such restrictions, then it can be used as a platform to freely distribute any content to anyone that wants it [37].

Further effort must go into exploring whether or not it is reasonable to have the P2P network enforce access control, or if the enforcements should take place at the endpoints of the network. In either case, only users that own (or have paid for) the right to download and access certain files should be able to do so to legally support data sharing applications.

If the benefits of P2P systems are to be realized, we need to explore the feasibility of and the technical approaches to instrumenting them with appropriate mechanisms to allow for the management of intellectual property.

## 4 Conclusion

Many of the open problems in P2P data sharing systems surround search and security issues. The key research problem in providing a search mechanism is how to provide for maximum expressiveness, comprhensiveness, and autonomy with the best possible efficiency, quality-of-service, and robustness. The key to securing a P2P network lies in designing mechanisms that ensure availabiity, file authenticity, anonymity, and access control. In this paper, we have illustrated some of the trade-offs at the heart of search and security problems in P2P data sharing systems, and outlined several major areas of importance for future work.

## References

1. Gnutella website. http://www.gnutella.com
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. Proc. ACM SIGCOMM (2001)
3. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Proc. ACM SIGCOMM (2001)
4. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proc. of the 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms (2001)
5. Ratnasamy, S., Shenker, S., Stoica, I.: Routing Algorithms for DHTs: Some Open Questions. Proc. IPTPS (2002)
6. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. Proc. of Intl. Conf. on Supercomputing (2002)
7. Cuenca-Acuna, F. M., Peery, C., Martin, R. P., Nguyen, T. D.: PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities. Technical Report DCS-TR-487, Dept. of Computer Science, Rutgers Univ. (2002)
8. Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R.: Estimating the size of a peer-to-peer network (2002)
9. Harren, M., Hellerstein, M., Huebsch, R., Loo, B., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. Proc. IPTPS (2002)
10. Crespo, A., Garcia-Molina, H.: Routing indicies for peer-to-peer systems. Proc. 28th Intl. Conf. on Distributed Computing Systems (2002)
11. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer systems. Proc. 28th Intl. Conf. on Distributed Computing Systems (2002)
12. Yang, B., Garcia-Molina, H.: Designing a super-peer network. Proc. ICDE (2003)
13. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services (2002)
14. Malkhi, D., Nao, M., Ratajczak, D.: Viceroy: a scalable and dynamic emulation of the butterfly. Proc. PODC (2002)
15. Harvey, N., Jones, M., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: a scalable overlay network with practical locality properties (2002)

16. Napster website. http://www.napster.com
17. Morpheus website. http://www.musiccity.com
18. W3C website on Web Services. http://www.w3.org/2002/ws
19. Seti@Home website. http://setiathome.ssl.berkely.edu
20. DataSynapse website. http://www.datasynapse.com
21. Groove Networks website. http://www.groove.net
22. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet Indirection Infrastructure. Proc. SIGCOMM (2002)
23. Stanford Peers group website. http://www-db.stanford.edu/peers
24. Reputation research network home page. http://databases.si.umich.edu/reputations/
25. Iris: Infrastructure for resilient internet systems. http://iris.lcs.mit.edu/
26. Personal communication with Dan Boneh.
27. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Workshop on Design Issues in Anonymity and Unobservability, pages 46–66 (2000)
28. Cooper, B., Garcia-Molina., H.: Peer to peer data trading to preserve information. ACM Transactions on Information Systems (2002)
29. Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. Workshop on Design Issues in Anonymity and Unobservability, pages 67–95 (2000)
30. Freedman, M.J., Morris, R.: Tarzan: A peer-to-peer anonymizing network layer. Proc. 9th ACM Conference on Computer and Communications Security, Washington, D.C. (2002)
31. Garber, L.: Denial-of-service attacks rip the internet. Computer, pages 12-17 (April 2000)
32. Hill, R., Hwang, A., Molnar, D.: Approaches to mixnets.
33. Lethin, R.: Chapter 17: Reputation. Peer-to-Peer: Harnessing the Power of Disruptive Technologies ed. Andy Oram, O'Reilly and Associates (2001)
34. Lynch, N.A.: Distributed algorithms. Morgan Kaufmann (1996)
35. Maniatis, P., Baker, M.: Secure History Preservation Through Timeline Entanglement. Proc. 11th USENIX Security Symposium, SF, CA, USA (2002)
36. Ganesh, A., Rowstron, A., Castro, M., Druschel, P., Wallach, D.: Security for structured peer-to-peer overlay networks. Proc. 5th OSDI, Boston, MA (2002)
37. Peinado, M., Biddle, P., England, P., Willman, B.: The darknet and the future of content distribution. http://crypto.stanford.edu/DRM2002/darknet5.doc.
38. Daswani, N., Garcia-Molina, H.: Query-flood DoS Attacks in Gnutella. Proc. Ninth ACM Conference on Computer and Communications Security, Washington, DC (2002)
39. Molnar, D., Dingledine, R., Freedman, M.: Chapter 12: Free haven. Peer-to-Peer: Harnessing the Power of Disruptive Technologies ed. Andy Oram, O'Reilly and Associates (2001)
40. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for Web transactions. ACM Transactions on Information and System Security, 1(1):66–92 (1998)
41. Resnick, P., Zeckhauser, R., Friedman, E., Kuwabara, K.: Reputation systems. Communications of the ACM, pages 45–48 (2000)
42. Shamir, A.: How to share a secret. Communications of the ACM, 22:612–613 (1979)
43. Sit, E., Morris, R.: Security considerations for peer-to-peer distributed hash tables. IPTPS '02, http://www.cs.rice.edu/Conferences/IPTPS02/173.pdf (2002)
44. Wright, M., Adler, M., Levine, B., Shields, C.: An analysis of the degradation of anonymous protocols. Technical Report, Univ. of Massachusetts, Amherst (2001)

# Approximations in Database Systems*

Yannis Ioannidis

Dept. of Informatics and Telecommunications, University of Athens, Hellas (Greece)
yannis@di.uoa.gr
http://www.di.uoa.gr/~yannis

**Abstract.** The need for approximations of information has become very critical in the recent past. From traditional query optimization to newer functionality like user feedback and knowledge discovery, data management systems require quick delivery of approximate data in order to serve their goals. There are several techniques that have been proposed to solve the problem, each with its own strengths and weaknesses. In this paper, we take a look at some of the most important data approximation problems and attempt to put them in a common framework and identify their similarities and differences. We then hint on some open and challenging problems that we believe are worth investigating.

## 1 Introduction

One of the main selling points of traditional database systems is consistency and accuracy. Given a database, the semantics of a query are precisely defined, and its answer is unique and exact. Approximations have been traditionally used only internally, as part of query optimization, whose goal is not really to find the actual optimal plan to execute a query but one that is close to it, avoiding the really poor ones. Given this approximate goal, database systems obtain approximations of the values of various size-related and cost-related parameters and use them to optimize queries.

In the recent past, however, things have changed and the need for approximations has increased sharply, moving also to the external layers of the database system. The answers to the queries themselves may now be approximate sometimes, as a quick feedback mechanism to the user on what to expect or because an accurate answer is expensive to obtain and not useful [5]. Furthermore, queries are no longer the only form of interaction with data, with on-line analytical processing, data mining, and other knowledge discovery methods playing a prominent role, all requiring essentially the extraction of approximate information from the database [2].

As a result, there has been extensive work on various aspects of the topic, with many interesting results. This work includes: extensive generalizations of earlier

* This is an approximate title as it does not quite represent the text that follows, which touches upon some issues that are beyond approximations or databases as well. The text itself is also an approximation, as it ignores several aspects of the topic, focusing only on personal curiosities.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 16–30, 2003.

database techniques for traditional (relational) data; invention of new techniques that approximate more complex types of data, e.g., spatial and temporal data; cross-fertilization with other areas that deal with information approximation, e.g., signal processing, and very successful adaptations of their techniques to the database environment (e.g., wavelets); redefinition of traditional problems of other areas (e.g., pattern recognition, statistics) and invention of scalable solutions that are effective when applied on large databases; and others.

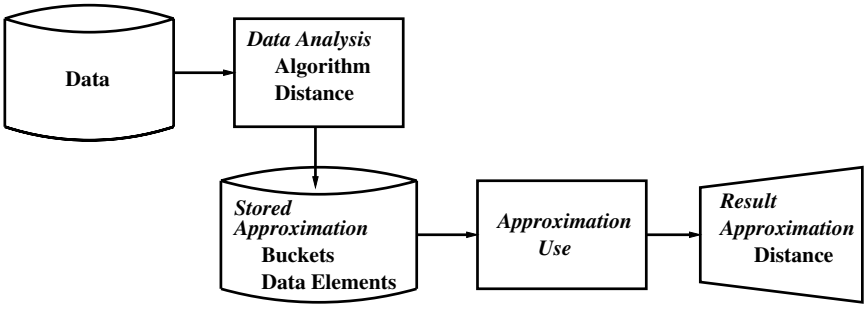



**Fig. 1.** Generic Flow of Approximation Process

For the purposes of this paper, any process of data approximation, exhibits roughly the flow shown in Figure 1. The nodes in that chain are described below:

1. *Data*: This is the original data that requires approximation.
2. *Data analysis*: This is the overall approach followed to derive an approximation of the incoming data, e.g., histograms or wavelets. Typically, incoming data elements are partitioned into groups of similar elements, called *buckets* (a term we use in this paper), *clusters*, *patterns*, and several other names. The data elements that fall in each bucket are then represented by the same approximate information. There are two aspects of each approximation technique that play a major role in its behavior:
   a) *Algorithm*: This is the algorithm run on the incoming data to generate its approximation.
   b) *Distance*: This is a measure of similarity between data elements that permits data elements that are "close" to be placed in the same bucket.
3. *Stored approximation*: This is the information generated by the approximation algorithm. It approximates the original data and is typically much smaller in size that it. This information is in the form of buckets representing parts of the space from which the original data is drawn. Again, there are two critical aspects of each bucket:
   a) *Bucket definition*: This is the specification of the part of the space that corresponds to a bucket.
   b) *Bucket data elements*: This is the information that is used to approximate the actual data elements that fall in a bucket.

4. *Approximation use*: This is the process by which the appropriate stored approximations are used to generate any approximate information required. Essentially, this corresponds to the actual purpose of the whole data approximation process.
5. *Result approximation*: This is the output of the previous step, i.e., the approximate information required at the end. A key parameter in characterizing this output is the following: that determines
   a) *Distance*: This is a measure of the quality of the generated approximate output, capturing its similarity to the corresponding output that would have been generated by applying a precise procedure on the original data. Usually it is intimately related to the distance parameter used by the data analysis performed earlier.

In this paper, we take a look at some of the most important data approximation processes that have been defined within different scientific areas and put them side by side with some of the most prominent techniques that have been proposed to solve them. We then hint on some open problems that we believe are interesting and challenging, whose solution may have some impact.

We should emphasize at this point that the entire area of data approximation is quite extensive and we make no attempt to be comprehensive or well-balanced by any means. We focus mostly on discrete data as it represents the primary concern of most data applications. Furthermore, we pay special attention to histograms and discuss them in more detail as they represent the main approximation technique used in current database systems. Finally, the choice of open problems is driven mostly by personal observations.

The paper is organized as follows. Section 2 defines data distributions and their characteristics that are useful for the rest of the paper and gives the essentials of histograms. Sections 3 to 8 outline some of the open problems that we consider worth investigating. Finally, Section 9 wraps up our thoughts.

## 2 Definitions

In the following, we briefly define several concepts on discrete data and present the key characteristics of histogram-based approximation, both of which are central in our discussion.

### 2.1 Data Distributions

For simplicity, we confine our detailed definitions on 1-dimensional data elements [9]. The *domain* $\mathcal{D}$ of an attribute $X$ in relation $R$ is the set of all possible values of $X$ and the (finite) *value set* $\mathcal{V}$ ($\subseteq \mathcal{D}$) is the set of values of $X$ that are actually present in $R$. Let $\mathcal{V} = \{ v_i \colon 1 \leq i \leq D \}$, where $v_i < v_j$ when $i < j$. The *spread* $s_i$ of $v_i$ is defined as $s_i = v_{i+1} - v_i$, for $1 \leq i < D$ (with $s_0 = v_1$ and $s_D = 1$)[1].

[1] The above holds for numerical attributes, where difference is defined. A commonly used technique for constructing histograms on non-numerical attributes (e.g., string

The *frequency* $f_i$ of $v_i$ is the number of tuples $t \in R$ with $t.X = v_i$. Finally, the *area* $a_i$ of $v_i$ is equal to $v_i \times f_i$. The *data distribution* of $X$ (in $R$) is the set of pairs $\mathcal{T} = \{ (v_1, f_1), (v_2, f_2), \ldots, (v_D, f_D) \}$.

The above can be extended to $n$-dimensional data elements, coming from multiple attributes $X_i, 1 \leq i \leq n$ of $R$ [8]. The *joint frequency* of a combination of $n$ values, one from each attribute, is the number of tuples in $R$ that contain the $i$-th value in attribute $X_i$, for all $i$. The *joint data distribution* $\mathcal{T}_{1,..,n}$ of $X_1, .., X_n$ is the entire set of *(value combination, joint frequency)* pairs.

### 2.2 Histograms

A *histogram* on an attribute $X$ is constructed by using a *partitioning rule* to partition the data distribution of $X$ into a number of mutually disjoint subsets called *buckets* and approximating the frequencies and values in each bucket in some common fashion. With respect to bucketization, several partitioning rules have been proposed. The most effective ones seem to be V-optimal(V,A) or V-optimal(V,F), whose buckets correspond to non-overlapping regions of $X$ values with the minimum overall variance of the areas (A) or the frequencies (F) of the values grouped together, respectively. The MaxDiff(V,A) and MaxDiff(V,F) histograms are almost as effective but are much simpler to construct [7], and therefore, maybe slightly more popular. Their buckets correspond to non-overlapping regions of $X$ values such that bucket breaks occur where the differences between the areas (A) or the frequencies (F), respectively, of neighboring values are maximized [9].

With respect to the information stored within each bucket to approximate the data elements that fall in it, the most common approach for values is to employ the *uniform spread assumption* [9], under which attribute values are assumed to be placed at equal intervals between the lowest and highest values in the bucket. Likewise, the most effective approach for frequencies is to employ the *uniform frequency assumption*, under which the frequencies in a bucket are approximated by their average.

Histograms can also be constructed on joint data distributions in a similar fashion. The task of identifying the appropriate partitioning of the multi-dimensional space in buckets is harder in this case, but several approaches have been developed that are quite successful in generating effective multi-dimensional histograms, at least for small numbers of dimensions [8,3].

## 3 Unification of Approximation Problems

In the past, several approximation problems have been defined that appear to have much similarity, yet they have been given different names and are approached using different techniques. Although it is clear that their differences

attributes) is to use a function that converts these vlaues into floating point numbers before constructing a histogram. For example, this technique is used in DB2.

are critical enough that they cannot all be unified completely, even partial unification would be really beneficial and could enrich the set of techniques applied to each of these problems significantly.

To illustrate the issues, we present three such problems and discuss their similarities and differences. One of them comes from the database world: *selectivity estimation*. The others come from the world of statistics and artificial intelligence: *clustering* and *classification*. We present all these techniques for the 2-dimensional case. With respect to the incoming 'Data' in Figure 1, we ignore any common differences in the focus of various problems and take a database perspective, i.e., we assume data elements on discrete numeric dimensions. Each of the other components of Figure 1 is discussed for each problem separately.

**Selectivity Estimation**

- *Data Analysis*: Partition the joint data distribution $\mathcal{T}_{1,2}$ into buckets, where each bucket contains similar elements. Similarity is defined based on some distance function that takes into account both the values of the two attributes and the value of the frequency.
- *Approximation Storage*: For each bucket, store a very short approximation of the $\mathcal{T}_{1,2}$ elements that fall there, typically by approximating each attribute/dimension and the frequency separately.
- *Approximation Use*: At query time, use the stored aggregate information to approximate the original $\mathcal{T}_{1,2}$ elements of the bucket, so that estimations of the query result size or sometimes the query result itself can be derived.

**Clustering**

- *Data Analysis*: Partition the joint data distribution $\mathcal{T}_{1,2}$ into buckets, where each bucket contains similar elements. Similarity is defined based on some distance function that takes into account just the values of the two attributes usually. Typically, clustering is used in cases where the frequency of each element is 1, but this does not have to be the case necessarily.
- *Approximation Storage*: For each bucket, store a very short approximation of the pairs of $\mathcal{V}_1 \times \mathcal{V}_2$ that fall there and the general area they occupy. Typically, this is a representative (not necessarily existing) element and a radius around it [10], but for scalable approaches, this may be much richer[2].
- *Approximation Use*: At insertion time, the inserted element is compared against the representative of each cluster and is placed in the most appropriate one (if any).

**Classification.** In this, next to every element of $\mathcal{V}_1 \times \mathcal{V}_2$ there is another attribute that takes values from a known, finite (typically small) set, the set of classes/categories. The goal is to identify the characteristics of the elements that fall into each class.

- *Data Analysis*: Partition the joint data distribution $\mathcal{T}_{1,2}$ into buckets, where each bucket contains similar elements. Similarity is defined based on some

distance function that takes into account just the values of the two attributes and on equality when it comes to the class attribute. Here again, in classification, the frequency of each element is 1. Even if it is not, however, it is mostly ignored and the effort concentrates on the pairs in $\mathcal{V}_1 \times \mathcal{V}_2$ (actually, usually on only one of the dimensions in each step of the algorithm) and the class attribute.

- *Approximation Storage*: For each bucket, store a very short approximation of the subspace occupied by the pairs of $\mathcal{V}_1 \times \mathcal{V}_2$ that fall there (but not the pairs themselves), typically through specifications of the borders of the subspace, as well as the category they represent.
- *Approximation Use*: At insertion time, the inserted element is compared against the areas of the buckets and is classified into the class associated with the bucket it falls in.

Note that the approximation use is rather distinct for each problem, and this is natural as to a large extent it defines the problem. For example, note that the purpose of selectivity estimation is dealing with queries, whereas the remaining problems deal mostly with updates (what happens when a new element arrives). On the other hand, often clustering may have no explicit use phase (the interest being in identifying clusters in existing information), whereas the other problems do for the most part.

With respect to the other two main components of Figure 1, however, comparing the three problems above and approaching them from a database perspective, we observe some striking similarities. The data analysis performed under each problem is essentially the same, yet the techniques that are used are very different. More or less the same holds for approximation storage. Clearly, some of these differences are to be expected, e.g., the presence of the additional class attribute for classification may have a positive or negative impact on the appropriateness of a specific analysis technique. In general, however, there appears to be no well documented reasoning for many of these differences.

The above raises several interesting questions. Why can't the histogram techniques that have been developed for selectivity estimation be used for clustering or vice versa? What would the impact be of using stored approximations developed for one problem to solve another? Is the different applicability of the techniques due to the differences in the intended use in each approximation problem? In that case, what is the extent of any such dependency and how can it be articulated? In general, given the great variety of techniques that exist for data approximation, it is crucial to obtain an understanding of the advantages and disadvantages of each one, its range of applicability, and in general, their relative characteristics when mutually compared. A comprehensive study needs to be conducted that will include several more techniques than those mentioned here. The "New Jersey Data Reduction Report" [1] has examined many techniques and has produced a preliminary comparison of their applicability to different types of data. It can serve as a good starting point for verification, extrapolation, and further exploration, not only with respect to applicability,

but also precise effectiveness trade-offs, efficiency of the algorithms, and other characteristics.

There are also questions on the nature of the approximation problems as well. Why can't the class attribute in classification be considered as an additional dimension of the data element space and have the problem be considered as clustering? When is this meaningful? Likewise, why can't the frequency in selectivity estimation be considered as another dimension of the joint data distribution and have the problem be considered as traditional clustering? The only distinction of the frequency is that there is a functional dependency from the attributes $X_1$ and $X_2$ to the frequency. What exactly does one gain (if anything) by taking advantage of that fact and treating frequency separately?

Elaborating on the last point, consider Figure 2 showing a typical data distribution of a single attribute $X$ of a relation. Assuming any of the established

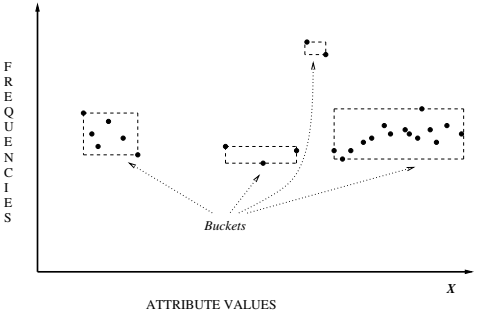



**Fig. 2.** One-Dimensional Data Distribution

most effective types of histograms mentioned earlier, the data distribution elements will be partitioned so that there is no overlap in the values of $X$ among the buckets and that similar frequencies and distances are placed in each bucket. Assuming further that there is space for four buckets, the buckets shown in Figure 2 are the likely outcome of any histogram construction algorithm.

Let us ignore, for a moment, the fact that the vertical axis represents frequencies and treat it as just a second dimension of the objects. The problem immediately changes to that of clustering; furthermore, applying any of the main clustering techniques that exist in the literature, it is very likely that the resulting clusters would have been the same. It is this relationship and similarity between the two problems that needs to be investigated and capitalized upon.

Such investigation is certainly not trivial and would need to address several general and detailed issues that arise. To mention just one of these issues, in Figure 2, one could argue that typical clustering algorithms applied on the entire 2-dimensional space might not have formed the second bucket. The reason would be that, although the y-values of the points (frequencies) are similar, the x-values are relatively far apart. The reason why histograms would probably form that bucket is that in selectivity estimation, the approximation of the x-axis in each

bucket is through some version of the uniform spread assumption. Hence, it is not proximity of the values themselves that is required for forming a bucket but proximity of their spreads. A hasty conclusion from the above may be that the selectivity estimation problem reduces precisely to clustering if applied on the space where the x-axis represents spreads instead of values. This is not quite true, however, as the spreads of the first points in every bucket are immaterial, so a traditional clustering algorithm would not succeed completely either. What one could do in this case is open for investigation.

As a last point of this section, we would like to note that there may be close relationships between the problems not only at the full-scale level discussed so far, but at a finer level as well. As a simple example, we consider the classification problem and one of its most important classes of algorithms: *decision tree* methods [2]. Operating on a multi-dimensional joint data distribution $\mathcal{T}_{1,..,n}$, the most popular decision tree algorithms (*ordinary binary classification trees*) sequentially split one of the dimensions in two pieces, eventually partitioning the space into hyper-rectangles whose elements belong to a unique class. Ignoring the tree-pruning aspects that these algorithms employ, which are irrelevant to our discussion, this is exactly how, for example, the MHIST approach to multi-dimensional histograms operates [8]. The splitting criteria employed in decision trees methods are different from those associated with MHIST, which are the traditional ones applied for 1-dimensional histogram construction. Could these be appropriate for decisions trees as well (applied on the class attribute)? Would the decision tree construction criteria be appropriate for histograms? Would some of the other histogram approaches be appropriate as a general approach to decision tree construction? This and several other potential relationships need to be worked on for the possible benefit of all problems.

## 4 Pattern (Bucket) Recognition

In all three problems mentioned above, data analysis is applied on a given set of dimensions. In traditional pattern recognition, however, there is usually an earlier stage where the dimensions are chosen among a great number of possibilities. A critical problem then is which dimensions to choose. There are several techniques that exist that address this problem with varying success depending on the case.

In selectivity estimation, dimension selection does not exist for the most part, as what needs to be approximated is usually known and given. Nevertheless, this issue may arise when we try to transform selectivity estimation into a clustering problem, as mentioned in Section 3, where the actual attribute values may have to be replaced by their corresponding spreads or something else. How does one choose appropriate alternatives? Although for the particular case, this may turn out to be not very hard, in general deriving dimensions in which patterns emerge, not by choosing among alternatives but by transforming them, is a very hard problem, not only for database estimation but for pattern recognition in general.

As an example, consider the 2-dimensional space above (whether the y-axis represents frequencies or not is irrelevant). It is clear to the human eye that

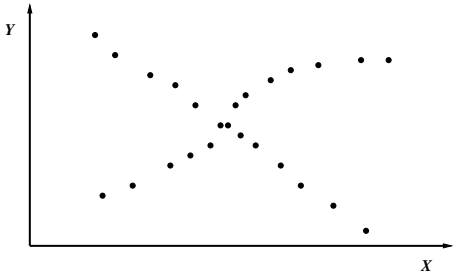



**Fig. 3.** Hard-to-Discover Patterns

there are two patterns in the figure, one with $Y$ decreasing and one with $Y$ increasing as $X$ increases. Nevertheless, the patterns cross each other in the $X$-$Y$ space, the elements that belong to each pattern do not satisfy any proximity criteria in that space, so current clustering techniques will most likely fail to identify them. One would have to model the elements in a very specific way, in a different multi-dimensional space, for the elements of each pattern to be close to each other and far from the elements of the other pattern. Understanding the mechanisms of such general pattern recognition is a great challenge that needs investigation. Such an effort should certainly try to address all five Gestalt Rules for clustering: small distance (on which current clustering techniques are quite effective), similarity, completion, continuity (where the example of Figure 3 falls), and symmetry. Some are harder to define than others, but we believe that an investigation in this direction will produce very interesting results.

## 5 Approximation within a Bucket

### 5.1 Approximation of Dimension Values and Frequencies

Consider the 1-dimensional data distribution of Figure 2, and the four buckets that have been indicated there. As mentioned in Section 2.2, a histogram makes certain uniformity assumptions within each bucket, most often the *uniform distribution assumption* for frequencies and some version of the *uniform spread assumption* for values. Hence, for the example of Figure 2, the stored information would generate the approximate distribution of Figure 4. Observe that frequencies are treated differently from attribute values: the frequencies in a bucket are assumed constant, whereas the values in a bucket are assumed to follow a linear rule. Hence, buckets need to store more information to approximate the values that fall in them than their frequencies. In principle, however, not all data distributions are served best with such an approach. For example, compare the three distributions of Figure 5. The first two are very similar cases, only that one can be thought of as rotated 90 degrees compared to the other. The third one is a mixture of the two. The current approach of histograms would work very

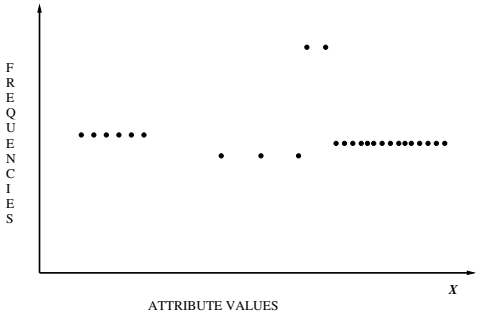



**Fig. 4.** Approximate One-Dimensional Data Distribution

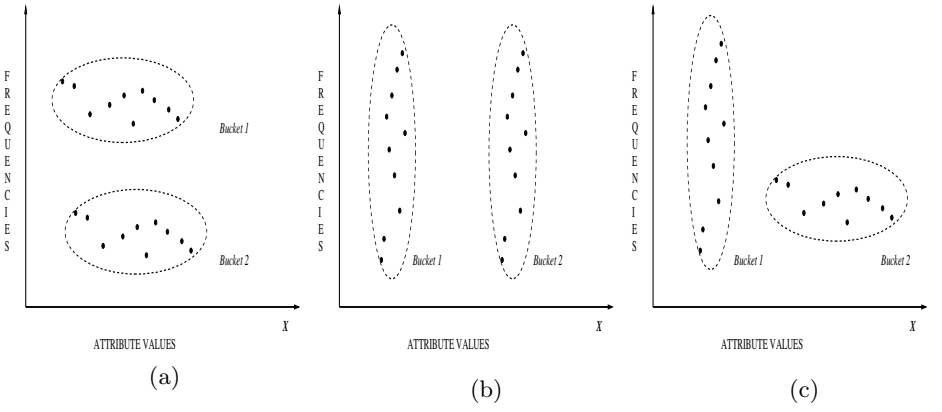



**Fig. 5.** (a) "Horizontal" Buckets, (b) "Vertical" Buckets, (c) "Mixed" Buckets

well with the left distribution[2], but not with the other two. For that second case, a much better approximation would be to store the average attribute value of a bucket (since they are all roughly the same) and assume a linear function for the frequencies that these roughly-equal values have. Likewise, for the third case, one bucket should be treated like the first case, and the other like the second case.

One may argue that range queries on values would perform poorly if all similar values are represented by their average. However, it is unclear if that would have a negative effect on the average performance if the values in a bucket come from a small range. One may further argue that distributions like those of Figure 5 are not expected to be found in databases, so the problem would never arise in practice. However, the problem remains even if the y-axis does not represent frequencies but a second value dimension, a case which is more

[2] This is assuming that buckets are allowed to overlap in the value domain, which is not typically the case, but this is orthogonal to the focus of this discussion which is approximation within a bucket.

plausible. In that case, current histograms would apply some version of the uniform spread assumption for both dimensions of all buckets, whereas clearly a distinct approach for each dimension in each bucket would have generated much better results.

To increase the accuracy of histogram approximations, there should be no fixed, predefined approximation approach to the value dimensions and the frequencies. Histograms should be flexible enough to use the optimal approximation for each dimension in each bucket, one that would produce the best estimations for the least amount of information. Identifying what that optimal approximation is is a hard problem and requires further investigation. Incidentally, note that this problem is intimately related to the problem of pattern identification mentioned above (Section 4), as identifying a pattern implies identifying a small amount of information that can be used to represent the elements of the pattern without much loss of information.

### 5.2 Approximation of Multi-dimensional Elements

Consider a 2-dimensional space and suppose that a 2-dimensional histogram uses the uniform spread assumption on each dimension of each bucket to approximate the values that appear there. The original values represent the projections of the actual elements that fall in the bucket, and the uniform spread assumption leads to an approximation of those projections. It does not offer an approximation for the actual placement of the elements in the 2-dimensional space. Current approaches make some gross oversimplifications to the problem, which may be responsible for large estimation errors. For example, if the projection of the elements on dimension $X_i$ generates $n_i$ unique values, $i \in \{1, 2\}$, then often the assumption is made that all elements of the cross-product of the values in the dimension projections are in the bucket. Clearly, this may be far from reality, as the cross-product has $n_1 * n_2$ elements, whereas the same projections might have been generated by as few as $max(n_1, n_2)$ actual elements.

The key question that arises then is how to take into account both the actual number of elements in a 2-dimensional (rectangular) bucket and their projections and place them uniformly in it. There does not seem to be an easy answer to this question. The source of the problem may be that the concept of uniformity is not necessarily appropriately defined in this case. Needless to say, the problem may be even harder in higher dimensions. We believe that any solution to this problem would improve the accuracy of histograms significantly in several cases.

## 6 Distance Metric

As shown in Figure 1, the concept of distance is central to approximation. It plays a major role in comparing data elements for bucket formation, identifying the appropriate approximate information to be stored, and even comparing results of data processing (on accurate and approximate data) for error measuring. Choosing the appropriate distance metric is crucial.

Clearly, there is no issue with the distance of numeric values, as there is a very natural definition of distance for them based on the difference of their actual values. The same is more or less true for elements in a multi-dimensional space with numeric dimensions, where there are several natural distance definitions (the $L$-norms) that produce useful results. The situation stops being as straightforward as soon as we move on to other types of elements, especially those with rich structure, e.g., graphs, XML files, and other semi-structured data. To illustrate the problem, we discuss two data types that are problematic: strings and sets (used mostly in the general sense of multisets, i.e., bags).

With respect to strings, a classical approach is to use the *edit distance*, i.e., the number of edit actions (insertions, deletions, replacements, swaps, etc.) that one has to perform in order to reach one string from the other. With respect to approximating string information in a bucket, a trie is usually employed. All these however, are only syntactic approaches to string distance. There are several cases, where a more semantic approach is warranted and might give better results. For example the strings 'city', 'town', and 'village' are very far apart in edit-distance, but are very close semantically. More often than not, the information conveyed by placing them in one bucket and using one of them to represent all three would be far more accurate than any other approach. What forms of semantic distances would make sense, how to maintain such knowledge that would allow for semantic distances to be calculated efficiently, and what the impact of using such distance metrics would be on approximations are questions that require further investigation.

With respect to sets, the problems are even harder. In comparing two sets, one has to overcome value mismatches (different member elements appearing in the two sets) and cardinality mismatches (both at the level of one set being larger than the other and at the level of one member element appearing different number of times in the two sets). The former could be consider as a special case of the latter using zero cardinality for the nonexistent elements, but it carries distinctly different semantics, so it should be treated separately. Traditional measures from other areas (e.g., Hausdorff distance) do not take into account either one of the mismatches above, hence, they are not very appropriate for approximations in databases. Other distances that have been introduced exactly for that purpose (e.g., DIST [6]) still seem to produce unintuitive distances on certain cases, e.g., the distance between $\{1, 1, 1, 2\}$ and $\{1, 2, 2, 2\}$ is equal to 12, the same as the distance between $\{1, 1, 1, 100\}$ and $\{1, 100, 100, 100\}$. Finding distance functions for sets that respect intuition and have useful mathematic properties (e.g., being metric) is quite a challenge, but would be a major step forward in approximating sets in databases.

## 7 Approximations and Indices

The fact that there is a close relationship between approximate statistics kept in databases, especially histograms, and indices has been recognized in the past in several works [1]. If one considers the root of a B+ tree, the values that

appear in it essentially partition the attribute on which it is built into buckets with the corresponding borders. Each bucket is then further subdivided into smaller buckets by the nodes of the subsequent level of the tree. One can imagine storing the appropriate information next to each bucket specified in a node, hence transforming the node into a histogram, and the entire index into a so called *hierarchical histogram*. This may adversely affect index search performance, of course, as it would reduce the out-degree of the node, possibly making the tree deeper. Nevertheless, although this idea works against the main functionality of an index, its benefits are non-negligible as well, so it has even been incorporated into some systems.

We believe that hierarchical histograms and, in general, the interaction between approximation structures and indices should be investigated further, as there are several interesting issues that remain unexplored as analyzed below. Consider again a B+ tree whose nodes are completely full, and assume for simplicity that it has been built on a foreign key of a relation. In that case, the root of the tree specifies a bucketization of the attribute domain that corresponds to an equi-depth histogram, i.e., each bucket contains an equal number of elements under it. Similarly, any node in the tree specifies an equi-depth bucketization of the range of values in leads to. If the nodes of the tree are not completely full, the equi-depth property does not quite hold, in the worst case one bucket's occupancy being within a factor o $2^h$ of another's, where $h$ is the depth of the tree. On the average, however, the equi-depth property should hold to a satisfactory level.

The main issue with B+ trees being turned into hierarchical equi-depth histograms is that equi-depth histograms are far from optimal overall on selectivity estimation [9]. Histograms like V-optimal(V,F), V-optimal(V,A), MaxDiff(V,F), and MaxDiff(V,A) are much more effective. What kind of indices would one get if each node represented bucketizations following one of these rules? Clearly, the trees would be unbalanced. This would make traditional search less efficient on the average. On the contrary, other forms of searches would be served more effectively. In particular, in a system that provides approximate answers to queries, the root of such a tree would provide a higher quality answer than the root of the corresponding B+ tree. Furthermore, the system may move in a progressive fashion, traversing the tree as usual and providing a series of answers that are continuously improving in quality, eventually reaching the leaves and the final, accurate result.

Returning to precise query answering, note that typically indices are built assuming all values or ranges of values being equally important. Hence, having a balanced tree becomes crucial. There are often cases, however, where different values have different importance and different frequency in the expected workloads. If this query frequency or some other such parameter is used in conjunction with advanced histogram bucketization rules, some very interesting trees would be generated whose average search performance might be much better than that of the B+ tree.

From the above, it is clear that the interaction between histograms and indices presents opportunities but also several technical challenges that need to be investigated. The trade-off between hierarchical histograms that are balanced trees with equi-depth bucketization and those that are unbalanced with more advanced bucketizations requires special attention. The possibility of some completely new structures that would strike even better trade-offs, combining the best of both worlds, cannot be ruled out either.

## 8 Original Data and Final Result Approximation

The last set of challenges we would like to mention are related to the two end components of Figure 1. As hinted in the rest of the paper, most current approximation efforts are applicable to data elements with numeric fields, whether alone in a 1-dimensional space or combined in multi-dimensional spaces. Very little has been done on categorical domains. Current practice is to map the categorical values onto a numeric domain and then use its properties. This often presents problems, as categorical values exhibit behavior that is unnatural to them. The earlier discussion on possible distance functions for strings may be relevant in general for categorical domains and their approximation. Even greater challenges are presented in approximating information with more complex structure, e.g., graphs or XML files.

On the other end, most current efforts on selectivity estimation have focused on selection queries. Dealing with joins or other operators or combinations of them has been rare [4] and has not produced completely satisfactory results. Given the importance of more complex queries, these problems should be given some special attention.

## 9 Conclusions

The importance of approximation in database systems will continue to grow, as increasingly more such functionality will be required. This will come both from demands for higher quality approximations in domains that are already being dealt with in database systems, e.g., better histograms, and from demands for new functionality, e.g., progressive refinement of query answers or advanced data mining applications.

We have identified several problems that we believe they represent nontrivial technical challenges and their solution (or any progress toward a solution) would improve our understanding of how information should be approximated most effectively and could influence directly or indirectly current approaches. These problems are summarized below:

- To what extent can the various approximation problems defined so far be unified under a common model, and what is the effectiveness of techniques developed for one such problem when used for another?

- What are the underlying principles of human pattern recognition and how can they be used to develop techniques for bucketizing data in the most effective way even when that is not a result of proximity in the natural space where the data resides?
- Given a set of elements that fall in a bucket, what information should be stored in the bucket for the most effective approximation of the set, in terms of obtaining the best trade-off between information size and approximation error?
- What are intuitive and mathematically robust distance functions for non-numeric data elements or data elements with complex structure?
- What are the opportunities that arise from combining index technology with approximation technology and how can the trade-off between efficient index searches and high-quality approximations be balanced?
- How can complex data elements and complex operator results be approximated?

It seems that there is a lot of work on approximations in front of us. Hopefully, as a community, we will be able to move forward on some of these problems, and have some fun on the way too!

**Acknowledgements.** We would like to thank Minos Garofalakis and Raghu Ramakrishnan for their comments on an earlier version of this paper. Their feedback was crucial in making this paper a better approximation of reality!

## References

1. Barbará D., et al.: The New Jersey Data Reduction Report. Data Engineering Bulletin **20:4** (1997) 3–45
2. Bradley P., Gehrke J., Ramakrishnan R., Srikant R.: Scaling Mining Algorithms to Large Databases. CACM **45:8** (2002) 38–43
3. Bruno N., Chaudhuri S., Gravano L.: STHoles: A Multidimensional Workload-Aware Histogram. SIGMOD Conference (2001) 294–305
4. Chakrabarti K., Garofalakis M., Rastogi R., Shim K.: Approximate Query Processing Using Wavelets. VLDB Journal **10:2-3** (2001) 199–223
5. Garofalakis M., Gibbons P.: Approximate Query Processing: Taming the Terabytes. VLDB Conference Tutorial (2001)
6. Ioannidis Y., Poosala V.: Histogram-Based Approximation of Set-Valued Query-Answers. VLDB Conference (1999) 174–185
7. Jagadish H. V., et al.: Optimal Histograms with Quality Guarantees. VLDB Conference (1998) 275–286
8. Poosala V., Ioannidis Y.: Selectivity Estimation Without the Attribute Value Independence Assumption. VLDB Conference (1997) 486–495
9. Poosala V., Ioannidis Y., Haas P., Shekita E.: Improved Histograms for Selectivity Estimation of Range Predicates. SIGMOD Conference (1996) 294–305
10. Theodoridis, S.: Pattern Recognition. Encyclopedia of Information Systems, Vol. 3. Elsevier Science (2003) 459–479

# Bioinformatics Adventures in Database Research

Jinyan Li, See-Kiong Ng, and Limsoon Wong

Laboratories for Information Technology
21 Heng Mui Keng Terrace, Singapore 119613
{jinyan, sknq, limsoon}@lit.a-star.edu.sg

**Abstract.** Informatics has helped launch molecular biology into the genomic era. It appears certain that informatics will remain a major contributor to molecular biology in the post-genome era. We discuss here data integration and datamining in bioinformatics, as well as the role that database theory played in these topics. We also describe LIMS as a third key topic in bioinformatics where advances in database system and theory can be very relevant.

## 1 The Great Challenges in Bioinformatics

Bioinformatics is key to genome projects. The main challenges are to improve the content and utility of databases, to develop better tools for data generation, capture, and annotation, to develop and improve tools and databases for functional studies, to develop and improve tools for representing and analysing sequence similarity and variation, and to create mechanisms to support effective approaches for producing robust software that can be widely shared. In the 1990s, a lot of databases were produced by the numerous mapping and sequencing initiatives. The hot topics were problems of managing and integrating these databases, and of comparing and assembling sequences contained in these databases. In the 2000s, several genomes have been completely sequenced and we enter the era of post-genome knowledge discovery. The hot topics are problems of recognizing specific features and functions of DNA and protein sequences, understanding their role in diseases, and their interaction with each other and their environment.

We focus on two stories. The first story concerns data integration challenges in bioinformatics in the 1990s. We narrate the triumph of the theory of query languages and the Kleisli system [6,39,9] in addressing these challenges. The second story concerns knowledge discovery challenges in bioinformatics in the 2000s. We narrate the tantalizing success of emerging patterns and the PCL technique [23,22] in addressing some of these challenges. This story is still an ongoing saga.

The paper is organized as follows. Section 2 presents the story of Kleisli. It begins with a discussion (Subsection 2.1) on what are the problems in the integrating biological data, and names (Subsection 2.2) the key features of a successful general solution to this problem. The key features are nested relational data model (Subsection 2.3), self-describing data exchange format (Subsection 2.4), thin wrapper (Subsection 2.5), and high-level query language (Subsection 2.6).

Section 3 presents the story of PCL. It begins with a discussion (Subsection 3.1) on knowledge discovery from biomedical data, and uses the recognition of translation initiation site from mRNA sequences as an example (Subsection 3.2). Then several general

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 31–46, 2003.

techniques for feature generation (Subsection 3.3) and feature selection (Subsection 3.4) are introduced. After that, PCL is introduced and applied to the recognition of translation initiation sites (Subsection 3.5). The algorithmic aspects underlying PCL are considered in Subsection 3.6.

Section 4 wraps up the paper with a discussion on laboratory information management systems or LIMS. We describe the key considerations for LIMS in the management of data and processes in a genome laboratory. While we have not conducted extensive research on this topic, we feel that LIMS is an area where advances in database system and theory, especially those pertaining to workflows, can offer significant benefit to molecular biology research.

## 2 From Theory of Query Languages to Biological Data Integration

### 2.1 Integration: What Are the Problems?

Many kinds of data are used in research in biology and medicine. They include laboratory records, literatures, nucleic acid and amino acid sequences, microarray output, and so on, as well as many computational tools for analysis of these data. We use the term data sources to refer to both databases and computational analysis tools. These data sources were mostly designed and built by biologists for their own individual use. This root leads to a significant legacy problem [2]. At the same time, it is realised that these data sources can also used in combination to answer questions that cannot be answered by any single data source [10]. This is thus a great challenge for the research and development of a general data integration system that can handle the heterogeneous, complex, and geographically dispersed nature of biomedical data sources. Let us introduce an "impossible query" from the US Department of Energy 1993 Bioinformatics Summit Report [10] that made this challenge famous.

*Example 1.* The query was to find, as many as possible, non-human homologs of each gene on a given cytogenetic band of a given human chromosome. Basically, this query means that for each gene in a particular position in the human genome, find DNA sequences from non-human organisms that are similar to it.

| source | type | location | remarks |
|---|---|---|---|
| GDB | Sybase | Baltimore | Flat tables, SQL joins, cytogenetic band info |
| Entrez | ASN.1 | Bethesda | Nested tables, keyword retrieveal, homolog info |

The challenge in this query lies not in the biology, but in the data sources as summarised in the table above. At the time it was posed, the main database containing cytogenetic band information was the GDB [29], which was a Sybase system. GDB did not store the DNA sequences that were needed for determining homologs. The sequences needed were kept in GenBank, which at that time could only be accessed via the ASN.1 version of Entrez [33]. Entrez also kept precomputed homologs. So this query needed the integration of GDB (a relational database located in Baltimore) and Entrez (a non-relational "database" located in Bethesda). This query was considered "impossible" as there was at that time no system that could work across the bioinformatics sources involved due to their heterogeneity, complexity, and geographical locations. □

### 2.2 Integration: Solution

A large number of solutions have been proposed to the data integration challenge in biomedicine. The main features of the most general and successful amongst these solutions are: data models that support nesting of structures, simple self-describing data exchange formats, thin wrappers, high-level query languages, good query optimizers, host language interfaces, and backend stores that support nesting of structures. The notions of data models, query languages, query optimizers, and backend stores are classic ideas in relational database management systems. Even the host language interface idea has manifested itself in the development of relational database systems as embedded SQL, JDBC and ODBC for quite some time. The arena of biomedical data integration requires that these ideas be extended to nested relations or other equivalent forms. These ideas still assume that the database system is the center of the world and that all data are kept within the database system. On the other hand, the ideas of data exchange formats and thin wrappers have appeared explicitly only in the mid 90s [28]. Furthermore, these ideas assume that data can be external to the database system and must be communicated and accessed in different ways.

It is by embracing these generalized old ideas of the database world and newer ideas outside of the database world that Kleisli [9] became the first successful general solution to the data integration problem in biomedicine. In the next subsections, we describe Kleisli and the influence of query language theory on its design and implementation.

### 2.3 Nested Relational Data Model

The nested relational data model was first proposed by Makinouchi [26] and enjoyed a fruitful evolution in a sequence of works [18,35] in the database world. However, the inspiration for the nested relational data model as in the Kleisli system arises from a different consideration [5]. In modern programming language theory, the key concepts are orthogonality and universality. These concepts together mean that for every data type, there should be constructs to build data of that type and constructs to take apart data of that type so that certain equations hold, and furthermore, so long as typing constraints are respected, these constructs can be freely combined. When the constructs for building sets and records are freely combined, nested sets arise naturally in Kleisli.

A data model is focused on the logical structure of data. The lexical aspect of data is irrelevant. The application semantic aspect of data is also irrelevant. A data model also comes with a query language for manipulating that logical structure. The link between the logical structure and application semantics of the data is captured by queries that are written a query language. The link between the logical structure and the lexical structure of the data is captured by a data exchange format. We discuss these links next.

### 2.4 Self-Describing Data Exchange Format

The concept of a self-describing data exchange format is implicit in programming languages that support type inference [8]. In these programming languages, every linguistic construct has an unambiguous meaning and type. The linguistic constructs for building data thus gives a self-describing data exchange format. The same concept arises explicitly, but much later, in database research [28].

The aspects of a self-describing data exchange format are: "lexical"—how to parse? "logical"—what is the structure? and "semantics"—what is the meaning? The lexical and logical layers are the important aspects of an exchange format. The better they are designed, the easier it is to exchange information. The semantics layer is not needed to accomplish the exchange of data between two systems. In addition, the lexical layer should support multiple logical layers. That is, objects of different structures can be laid out in a data stream using the same lexical layer. "Printing" and "parsing" are the two operations that map between the lexical and logical layers. "Transmit" is the operation to move data laid out in the data exchange format between two systems. All good data exchange systems provide the equivalent of "print", "transmit", and "parse".

*Example 2.* We first present a poor data exchange format. Here is a GenPept report of a F-box protein. The GenPept report is an inconvenient data exchange format as its logical and lexical layers are not explicit. It is also not a versatile data exchange format as it cannot be used for any other kind of data.

```
LOCUS       T41727        577 aa        PLN       03-DEC-1999
DEFINITION  F-box domain protein Pof3p - fission yeast
ACCESSION   T41727
PID         g7490551
VERSION     T41727  GI:7490551
DBSOURCE    pir: locus T41727;
            summary: #length 577 #weight 66238 ...
KEYWORDS    .
SOURCE      fission yeast.
  ORGANISM  Schizosaccharomyces pombe
            Eukaryota; Fungi; Ascomycota; ...
REFERENCE   1  (residues 1 to 577)
  AUTHORS   Lyne,M., Wood,V., Rajandream,M.A., ...
  TITLE     Direct Submission
  JOURNAL   Submitted (??-JUN-1998) to the EMBL Data Library
FEATURES             Location/Qualifiers
     source          1..577
                     /organism="Schizosaccharomyces pombe"
                     /db_xref="taxon:4896"
     Protein         1..577
                     /product="F-box domain protein Pof3p"
ORIGIN
   1 mnnyqvkaik ektqqylskr kfedaltfit ktieqepnpt ...
```

□

The table below is a better exchange format used in Kleisli [40]. The logical and lexical layers are separated so that a generic "parser" can be used for any data laid out in this exchange format. The logical layer conforms to the nested relational model. This allows a generic parser to be built into Kleisli so that it can directly parse and manipulate any data conforming to this exchange format. Also, Kleisli outputs in this format by default.

| logical layer | lexical layer | remarks |
|---|---|---|
| Booleans | true, false | |
| Numbers | 123, 123.123 | Positive numbers |
| | –123, –123.123 | Negative numbers |
| Strings | "a string" | String is inside double quotes |
| Records | (#$l_1$: $O_1$, ..., #$l_n$: $O_n$) | Record is inside round brackets |
| Sets | { $O_1$, ..., $O_n$} | Set is inside curly brackets |
| Lists | [ $O_1$, ..., $O_n$] | List is inside square brackets |

*Example 3.* The F-box report from Example 2 looks like this in the data exchange format used by Kleisli:

```
(#uid: 7490551,
 #title: "F-box domain protein Pof3p - fission yeast",
 #accession: "T41727",
 #common: "fission yeast.",
 #organism: (#genus: "Schizosaccharomyces",
        #species: "pombe",
        #lineage: ["Eukaryota", "Fungi", "Ascomycota", ...]),
 #feature: {(#name: "source", #start: 0, #end: 576,
        #anno: [(#anno_name: "organism",
                 #descr: "Schizosaccharomyces pombe"),
                (#anno_name:"db_xref", #descr:"taxon:4896")]),
       (#name: "Protein", #start: 0, #end: 576,
        #anno: [(#anno_name: "product",
                 #descr: "F-box domain protein Pof3p")])},
 #sequence: "MNNYQVKAIKEKTQQYLSKRKFEDALTFITKTIEQEPNPTID...")
```

This format change is simple and has a few positive consequences: (a) the boundaries of different nested structure becomes explicit, and (b) the lexical and logical aspects are no longer mixed up. As a result, a parser specific to the GenPept format is not needed. □

### 2.5 Thin Wrappers

A wrapper is an interface that (a) conveys requests from one system to a second system, (b) conveys replies from the second system to the first system, and (c) performs any necessary lexical/logical mapping. Whether a wrapper is easy to implement or not is affected by the following factors: the impedance between the data models of the two systems, the complexity of the access protocols of the two systems, and the suitability of the programming language chosen for implementing the wrapper.

As most of the data sources are flat files, the complexity of the access protocol for them is low. In systems such as Kleisli that are based on a nested relational data model, the data model impedance is also low. In systems such as Kleisli that has a simple data exchange format—which also conforms to the nested relational data model—the complexity of the protocol required for conveying replies to them is low. These two advantages for wrapper development in the Kleisli system can be attributed to its strong query language theory foundation [39]. Its nested relational data model is motivated by orthogonality considerations in query language design and its exchange format follows from type inference considerations in query language design.

In contrast, writing wrappers for a system like IBM's DiscoveryLink [16] requires more effort. DiscoveryLink relies on the flat relational data model, which causes high impedance mismatch with most of the biological data sources as these sources generally have nested structures. DiscoveryLink also lacks a simple data exchange format, which causes high complexity to get replies into the system. DiscoveryLink also insists on wrappers being written in C or C++, which are not convenient for parsers.

### 2.6 High-Level Query Language

Kleisli supports two high-level query languages: CPL [39] and sSQL [6]. Both are nested relational query languages and have the same expressive power. The former was the original query language of the Kleisli system and was inspired by research on the connection between comprehension and monads [37,4]. The latter was added to the Kleisli system more recently by popular demand and was based on SQL. Let us show the advantage of using a high level language like sSQL with the following query in sSQL: `select title from DATA`. All of the following low-level details that a Perl programmer has to handle explicitly—if the same query is implemented in Perl—are handled automatically by the Kleisli compiler for sSQL: input, lexical-logical mapping, type checking, termination, optimization, output, etc. These low-level details can be handled automatically because the right compromises have been made in sSQL that reduce its complexity and yet without hurting its ability to express queries important for large-scale data manipulations.

The core of CPL and sSQL has their roots in $\mathcal{NRC}$, a nested relational calculus. The syntax of $\mathcal{NRC}$ is summarised below and its semantics is given in earlier papers [5,38].

$$\frac{}{f_t^s : s \to t} \quad \frac{}{x^t : t} \quad \frac{e_1 : t \quad e_2 : t}{e_1 = e_2 : \mathcal{B}}$$

$$\frac{}{\mathit{true} : \mathcal{B}} \quad \frac{}{\mathit{false} : \mathcal{B}} \quad \frac{e : \mathcal{B} \quad e_1 : t \quad e_2 : t}{\mathit{if}\ e\ \mathit{then}\ e_1\ \mathit{else}\ e_2 : t} \quad \frac{e_1 : t_1 \quad e_2 : t_2}{(e_1, e_2) : t_1 \times t_2} \quad \frac{e : t_1 \times t_2}{\pi_1\, e : t_1} \quad \frac{e : t_1 \times t_2}{\pi_2\, e : t_2}$$

$$\frac{e : t}{\{e\} : \{t\}} \quad \frac{e_1 : \{t\} \quad e_2 : \{t\}}{e_1 \cup e_2 : \{t\}} \quad \frac{}{\{\}^t : \{t\}} \quad \frac{e_1 : \{t_1\} \quad e_2 : \{t_2\}}{\bigcup\{e_1 |\, x^{t_2} \in e_2\} : \{t_1\}}$$

The design of $\mathcal{NRC}$ is such that for every data type that it supports, it provides constructs for building data of that type and constructs for taking apart data of that type in a way that satisfies universality and orthogonality conditions. For the record or pair type: $(e_1, e_2)$ forms a pair; and $\pi_1\ e$ and $\pi_2\ e$ extracts the first and second component of the pair $e$. For the set type: $\{\}$, $\{e\}$, and $e_1\ \cup\ e_2$ form the empty set, the singleton set, and the union of two sets respectively; and $\bigcup\{e_1 \mid x \in e_2\}$ is the big union of $g(o_1)$, ..., $g(o_n)$ if $g(x) = e_1$ and $\{o_1, ..., o_n\} = e_2$. Also, $f$ stands for any additional functions that $\mathcal{NRC}$ can be extended with later. We denote extension to $\mathcal{NRC}$ in brackets. For example, $\mathcal{NRC}(\mathcal{Q}, +, \cdot, -, \div, \sum, \leq^{\mathcal{Q}}, =)$ means $\mathcal{NRC}$ extended with rational numbers, arithmetics, summation, and the linear order on rational numbers.

**Theorem 1 (Cf. [5,38,24]).**

1. $\mathcal{NRC}$ $=$ *Thomas&Fischer* $=$ *Schek&Scholl* $=$ *Colby.*
2. $\mathcal{NRC}$ $= FO(=) =$ *RelationalAlgebra over flat relations.*

*3.* $\mathcal{NRC}(\mathcal{Q}, +, \cdot, -, \div, \sum, \leq^{\mathcal{Q}}, =)$ *over flat relations is "entry-level" SQL.* □

These results say that $\mathcal{NRC}$ and its various extensions or restrictions, as the case may be, are equivalent to various classical query languages. They also bring out three important points. Firstly, $\mathcal{NRC}$ has enough expressive power for large-scale data manipulations commonly expected in database systems. Secondly, $\mathcal{NRC}$ can be straightforwardly extended by any number of functions $f : s \to t$ of any input-output type as shown in its syntax, whereas the classical query languages above are lacking in orthogonality in their design and cannot accomodate such extensions so easily. This ease of extension is a crucial when handling the integration of biomedical data sources that often include a lot of specialized access and analysis functions. Lastly, the proofs of these results all rely on a rewrite system based on the equation of monads. This rewrite system is also used as the starting point of Kleisli's query optimizer [39].

While $\mathcal{NRC}$ is amenable to automated analysis and optimization, it is not as amendable for a human to read. For example, the cartesian product of two sets is expressed in $\mathcal{NRC}$ as $\bigcup\{\bigcup\{\{(x,y)\} \mid y \in S\} \mid x \in R\}$. This is clumsy. Hence, we need an alternative surface syntax to be for users of the Kleisli system. A popular choice is sSQL, which is based on SQL and is very human readable. E.g., the same cartesian product is expressed as: `select (x,y) from R x, S y`. Queries in sSQL are converted into $\mathcal{NRC}$ via the Wadler equalities [37] linking monads and comprehensions. We present now the sSQL solution to Example 1.

*Example 4.*
```
sybase-add (#name:"GDB", ...);
create view L from locus_cyto_location using GDB;
create view E from object_genbank_eref using GDB;
select #accn: g.#genbank_ref, #nonhuman-homologs: H
from L c, E g,
  {select  u
   from g.#genbank_ref.na-get-homolog-summary u
   where not(u.#title string-islike "%Human%"),
         not(u.#title string-islike "%H.sapien%")} H
where c.#chrom_num = "22" andalso g.#object_id = c.#locus_id
 andalso not (H = { });
```
The first three lines connect to GDB and map two tables in GDB to Kleisli. These two tables could then be referenced within Kleisli as if they were two locally defined sets, `locus` and `eref`. The next few lines extract from these tables the accession numbers of genes on Chromosome 22, use the Entrez function `aa-get-homolog-summary` to get their homologs, and filter these homologs for non-human ones. □

We see that the integration of GDB and Entrez is smooth, and the handling of input, output, and lexical-logical mapping are fully taken care of. It is also very efficient. The Kleisli optimizer is able to migrate the join, selection, and projections on the two GDB tables into a single efficient access to GDB. The accesses to Entrez are also automatically made concurrent. These are just some advantages that a well-implemented system with a high-level query language can bring over a poor man's Perl library.

# 3 From Emerging Patterns to Biological Datamining

## 3.1 Prediction: What Are the Problems?

Microarrays are now being used to measure the expression level of thousands of genes simultaneously [14]. We can expect reasonably that gene expression data measured by microarrays or other means will soon be part of a patient's medical records. The analysis of medical records is aimed mainly at diagnosis, prognosis, and treatment planning. Here we are looking for patterns that are (a) valid: they are also observed in new data with high certainty; (b) novel: they are not obvious to experts and provide new insights; (c) useful: they enable reliable predictions; and (d) understandable: they pose no obstacle in their interpretation. In short, a classification technique that is both highly accurate and highly understandable is more desirable.

Gene expression profiles and medical records represent the spectrum of biomedical data that possess explicit signals or features—such as expression level of individual genes—where traditional machine learning and datamining techniques appear to be directly useable. However, these techniques have performance limitation in terms of the number of signals or features that they can handle in their input. A modern microarray such as the Affymetrix U95A GeneChip can produce a gene expression profile consisting over the expression levels of over 12000 genes. Such a large number of features is beyond what can be comfortably handled by most of these techniques. So techniques for dimension reduction or feature selection are also needed.

Not all biomedical data contain explicit signals or features. DNA sequences, protein sequences, and so on represent the spectrum of biomedical data that possess no explicit features. E.g., a genomic sequence is typically just a string consisting of the letters "A", "C", "G", and "T" in an apparently random order. And yet a genomic sequence possesses biologically meaningful structure points such as transcription start site, translation initiation site, and so on that are associated with the primary structure of genes. Recognition of these biological structure points in a genomic sequence is an important bioinformatics application. Traditional machine learning and datamining techniques cannot be directly apply to this type of recognition problems. So there is a need to adapt these existing techniques to this type of problems and a need for good techniques for generating explicit features underlying such structure points.

In the next subsections, we use the recognition of translation initiation site to illustrate the challenges above—feature generation, selection, and integration for prediction—of knowledge discovery in biological data.

## 3.2 Recognition of Translation Initiation Sites

Proteins are synthesized from mRNAs by a process called translation. The region at which the process initiates is called the Translation Initiation Site (TIS). The TIS recognition problem is to correctly identify the TIS in an mRNA or cDNA sequence. In Zien et al. [43] and Hatzigeorgiou [17], accurate TIS prediction is obtained by complex methods: A careful engineering of kernel functions for a support vector machine (SVM) and a multi-step integrated artificial neural network (ANN). Recently, one of us (Wong) shows that good performance can be obtained by simple feature generation and selection followed by a variety of standard machine learning methods [42]. In what follows,

we repeat this approach [42], but we use features generated from a translation of the mRNAs into the corresponding amino acids instead of the mRNAs directly, and we use PCL as the classification algorithm instead of traditional machine learning methods such as artificial neural network [32], Bayesian classifier [21], decision tree induction [31], support vector machine [36].

We use the vertebrate dataset provided by Pedersen and Nielsen [30]. This dataset was also used by Zien et al. [43]. These sequences are processed by removing possible introns and joining the exons [30] to obtain the corresponding mRNA sequences. From these sequences, only those with an annotated TIS, and with at least 100 upstream nucleotides as well as 100 downstream nucleotides are selected. The sequences are filtered to remove those belonging to same gene families, homologous genes from different organisms and sequences added multiple times to the database. Since the dataset is processed DNA, the TIS site is ATG. In total there are 13375 ATG sites. Of these possible ATG start sites, 3312 (24.76%) are the true start TIS, while the other 10063 (75.23%) are non-TIS. At each of these ATG sites, we extract a sequence segment consisting of (a) 100 nucleotides up-stream or upto the first occurrence of an ATG up-stream, whichever is shorter; and (b) 100 nucleotides down-stream. These 13375 sequence segments are the inputs upon which we perform the recognition of TIS.

### 3.3 Feature Generation

The sequences are not suitable for direct application of machine learning techniques, as these techniques rely on explicit signals of high quality. It is necessary to devise various sensors for such signals, so that given a sequence segment, a score is produced to indicate the possible presence of such a signal in that sequence segment. A general technique for producing these sensors is the idea of k-grams frequency. A k-gram is simply a pattern of k consecutive letters, which could be amino acid symbols or nucleic acid symbols. K-grams can also be restricted those in-frame ones. Each k-gram and its frequency in the said sequence fragment becomes a signal. Another general technique for producing these sensors is the idea of position-specific k-gram. The sensor simply reports what k-gram is seen in a particular position in the sequence fragment.

*Example 5.* For TIS prediction, we use a combination of the general techniques mentioned above. First, we divide each of our 13375 sequence segments into an up-stream part, i.e., to the left of the ATG, and a down-stream part, i.e., to the right of the ATG. Then we consider 3-grams that are in-frame, i.e., those 3-grams aligned to the ATG. Those in-frame 3-grams that code for amino acids are converted into amino acid letters. Those in-frame 3-grams that are stop codons are converted into a special letter symbolizing a stop codon. Then we generate the follow numeric features: `UP-X` (resp. `DOWN-X`), which counts the number of times the amino acid letter `X` appears in the up-stream part (resp. down-stream), for `X` ranging over the standard 20 amino acid letters. We also generate the following Boolean features: `UP-ATG`, which indicates an in-frame ATG occurs in the up-stream part; `DOWN-STOP` (resp. `UP-STOP`), which indicates a stop codon occurs in the down-stream part (resp. up-stream); and `UP`$n$`-N` (resp. `DOWN`$n$`-N`), which indicates the nucleotide `N` occurs in position $n$ up-stream (resp. down-stream), for `N` ranging over the 4 standard nucleotide letters (A, C, G, T) and 6 generalizations (AorC, AorG, AorT,

CorG, CorT, CorG), and for $n$ ranging over the 10 up-stream (resp. down-stream) positions closest to the ATG. Thus for each of our 13375 sequence segments, we have $(20 \times 2) + 1 + 2 + (10 \times 10 \times 2) = 243$ features. □

### 3.4 Feature Selection

As can be seen, the techniques described above can generate a large number of features for each sequence segment. Clearly, not every feature is important. Using only the more important features has the advantage of avoiding noise and speeding up subsequent construction of the recognition model. It is thus often desirable to discard weaker features. A number of general techniques can be used for this purpose [25,13].

Here, let us use the entropy method [13]. The basic idea of this method is to filter out those features whose value distributions are relatively random. For the remaining features, this method can automatically find some cut points in these features' value ranges such that the resulting value intervals of every feature can be maximally distinguished. If each value interval induced by the cut points of a feature contains only the same class of samples, then this partitioning by the cut points of this feature has an entropy value of zero. The smaller a feature's entropy is, the more discriminatory it is. Formally, let $T$ partition the set $S$ of examples into the subsets $S_1$ and $S_2$. Let there be $k$ classes $C_1, \cdots, C_k$. Let $P(C_i, S_j)$ be the proportion of examples in $S_j$ that have class $C_i$. The *class entropy* of a subset $S_j, j = 1, 2$ is defined as:

$$Ent(S_j) = -\sum_{i=1}^{k} P(C_i, S_j) log(P(C_i, S_j)).$$

Suppose the subsets $S_1$ and $S_2$ are induced by partitioning a feature $A$ at point $T$. Then, the *class information entropy* of the partition, denoted $E(A, T; S)$, is given by:

$$E(A, T; S) = \frac{|S_1|}{|S|} Ent(S_1) + \frac{|S_2|}{|S|} Ent(S_2).$$

A binary discretization for $A$ is determined by selecting the cut point $T_A$ for which $E(A, T; S)$ is minimal amongst all the candidate cut point. The same process can be applied recursively to $S_1$ and $S_2$. The *Minimal Description Length Principle* is used to stop partitioning. That is, recursive partitioning within a set of values $S$ stops iff

$$Gain(A, T; S) < \frac{log_2(N-1)}{N} + \frac{\delta(A, T; S)}{N},$$

where $N$ is the number of values in the set $S$, $Gain(A, T; S) = Ent(S) - E(A, T; S)$, $\delta(A, T; S) = log_2(3^k - 2) - [k \cdot Ent(S) - k_1 \cdot Ent(S_1) - k_2 \cdot Ent(S_2)]$, and $k_i$ is the number of class labels represented in the set $S_i$. We choose those features with lowest entropy values.

*Example 6.* Applying the entropy method to the features generated in Example 5 for the whole data set, the 10 features of lowest entropy are `UP-ATG`, `DOWN-STOP`, `DOWN-A`, `UP3-AorG`, `DOWN-V`, `UP-A`, `DOWN-E`, `DOWN-L`, `DOWN-D`, and `UP-G`. We also perform three-fold cross validation. The data set is divided into three groups of the same size, where

the proportion of true TIS to non-TIS is kept the same as the original undivided data set. In each fold, one of the groups is held out as test cases and the other two groups are used as training cases. In the training of each fold, we apply the entropy method to each group of training cases to select the 10 features of lowest entropy for that fold. It turns out that, for each fold, the same 10 features listed above are picked. Some of these 10 features also make good biological sense. The `UP-ATG` feature makes sense because it is uncommon for an in-frame up-stream ATG to be near a translation initiation site, as this runs counter to the scanning model of eukaryotic protein translation [19]. The `DOWN-STOP` feature makes sense because it is uncommon for an in-frame stop codon to be near a translation initiation site, as this implies an improbably short protein product. The `UP3-AorG` feature makes sense because it is consistent with the well-known Kozak consensus signature observed at translation initiation sites [19]. □

### 3.5 Prediction: Solution

In the field of machine learning, there are many good prediction methods including $k$-nearest neighbours ($k$-NN) [7], C4.5 [31], SVM [36], NB [21], etc. C4.5 induces from training data rules that are easy to comprehend. However, it may not have good performance if real decision boundary underlying the data is not linear. NB uses Bayesian rules to compute a probabilistic summary for each class. A key assumption used in NB is that the underlying features are statistically independent. However, this is not appropriate for gene expression data analysis as genes involved in an expression profile are often closely related and appear not to be independent. $k$-NN assigns a testing sample the class of its nearest training sample in terms of some non-linear distance functions. Even though $k$-NN is intuitive and has good performance, it is not helpful for understanding complex cases in depth. SVM uses non-linear kernel functions to construct a complicated mapping between samples and their class labels. SVM has good performance, but it functions as a black box. Similarly, traditional datamining methods that look for high frequency patterns are not useful on these data. E.g., if you use these methods in the Singapore General Hospital, they will produce totally useless patterns such as "everyone here has black hair and black eyes."

We are therefore motivated to seek a classification method that enjoys the twin advantages of high accuracy and high comprehensibility. We have been developing a novel classification method called PCL [23,22] for Prediction by Collective Likelihood of emerging patterns. This method focuses on (a) fast techniques for identifying patterns whose frequencies in two classes differ by a large ratio [11], which are the so-called emerging patterns; and on (b) combining these patterns to make decision. For PCL, we use only emerging patterns that are most general and have infinite ratio.

Basically, the PCL classifier has two phases. Given two training datasets $D^P$ (instances of class $P$) and $D^N$ (instances of class $N$) and a test sample $T$, PCL first discovers two groups of most general emerging patterns from $D^P$ and $D^N$. Denote the most general emerging patterns of $D^P$ as, $EP_1^P$, $EP_2^P$, $\cdots$, $EP_i^P$, in descending order of frequency. Denote the most general emerging patterns of $D^N$ as $EP_1^N, EP_2^N, \cdots, EP_j^N$, in descending order of frequency. Suppose the test sample $T$ contains these most general emerging patterns of $D^P$: $EP_{i_1}^P$, $EP_{i_2}^P$, $\cdots$, $EP_{i_x}^P$, $i_1 < i_2 < \cdots < i_x \leq i$, and these most general emerging patterns of $D^N$: $EP_{j_1}^N$, $EP_{j_2}^N$, $\cdots$, $EP_{j_y}^N$, $j_1 < j_2 < \cdots < j_y \leq j$.

The next step is to calculate two scores for predicting the class label of $T$. Suppose we use $k$ ($k \ll i$ and $k \ll j$) top-ranked most general emerging patterns of $D^P$ and $D^N$. Then we define the score of $T$ in the $D^P$ class as

$$score(T, D^P) = \sum_{m=1}^{k} \frac{frequency(EP^P_{i_m})}{frequency(EP^P_m)},$$

and the score in the $D^N$ class is similarly defined in terms of $EP^N_{j_m}$ and $EP^N_m$. If $score(T, D^P) > score(T, D^N)$, then $T$ is predicted as the class of $D^P$. Otherwise it is predicted as the class of $D^N$. We use the size of $D^P$ and $D^N$ to break tie. The PCL classifier has proved to be a good tool for analysing gene expression data and proteomic data [22,41,23]. Here we use it on recognizing translation initiation sites.

*Example 7.* We apply PCL, with $k = 10$, in the same 3-fold cross validation using the top 10 features selected in each fold as per Example 6. An accuracy of 87.7% is obtained as shown in the table below. This accuracy of 87.7% is comparable to the 89.4% in our earlier work [42] using a similar approach of feature generation and feature selection followed by the application of a machine learning method, except that our earlier work considered NB, SVM, and C4.5 on the same data set and used only nucleotide k-mers. It also compares well with the 78% sensitivity on TIS, 87% sensitivity on non-TIS, and 85% accuracy reported by Pedersen and Nielsen [30] on the same data set. It also compares favourably with the 69.9% sensitivity on TIS, 94.1% sensitivity on non-TIS, and 88.1% accuracy reported by Zien et al [43] on the same data set. Further improvement can be obtained by incorporating a form of distance or scanning rule [17,42].

| | 1st fold | 2nd fold | 3rd fold | overall |
|---|---|---|---|---|
| no. of TIS ($a$) | 1104 | 1104 | 1104 | 3312 |
| correct predictions ($b$) | 926 | 936 | 944 | 2806 |
| wrong predictions ($c$) | 178 | 168 | 160 | 506 |
| sensitivity ($a/b$) | 83.9% | 84.8% | 85.5% | 84.7% |
| no. of non-TIS ($d$) | 3355 | 3354 | 3354 | 10063 |
| correct predictions ($e$) | 2998 | 2928 | 2996 | 8922 |
| wrong predictions ($f$) | 357 | 426 | 358 | 1141 |
| sensitivity ($e/d$) | 89.4% | 87.3% | 89.3% | 88.7% |
| accuracy ($[b+e]/[a+d]$) | 88.0% | 86.7% | 88.4% | 87.7% |

□

Beyond good accuracy, PCL also has the advantage of producing comprehensible decision rules derived from the top emerging patterns. E.g., the emerging pattern $\{\texttt{UP} - \texttt{ATG} = Y, \texttt{DOWN} - \texttt{STOP} = Y, \texttt{DOWN} - \texttt{A} = 0\}$ has 10% support in our non-TIS sequence segments and 0% support in our TIS sequence segments. Then a rule can be derived from this emerging pattern as: *If the sequence segment has an in-frame ATG up-stream from the candidate TIS, and an in-frame stop codon down-stream from the candidate TIS, and there is no occurrence of any codons for the amino acid A, then this candidate TIS is a non-TIS with 10% probability and is a TIS with 0% probability.*

### 3.6 A Practical Challenge

Given two datasets $D^P$ and $D^N$, an emerging pattern—to be used by PCL—is a pattern such that its frequency in $D^P$ (or $D^N$) is non-zero but in the other dataset is zero (i.e., infinite ratio between its support in $D^P$ and $D^N$), and none of its proper subpattern is an emerging pattern (i.e., most general). If $D^P$ and $D^N$ have $n$ Boolean attributes, then there are $2^n$ possible patterns. Hence a naive method to extract emerging patterns requires $2^n$ scans of the dataset. A more efficient method for extracting emerging patterns is therefore crucial to the operation of PCL. Let us begin with a theoretical observation:

**Theorem 2 (Cf. [12]).** *The emerging patterns from $D^P$ to $D^N$—all of them together, not just most general the ones—form a convex space.* □

It is known that a convex space $C$ can be represented by a pair of borders $\langle L, R\rangle$, so that (a) $L$ is anti-chain, (b) $R$ is anti-chain, (c) each $X \in L$ is more general than some $Z \in R$, (d) each $Z \in R$ is more specific than some $X \in L$, and (e) $C = \{Y \mid \exists X \in L, \exists Z \in R, X \subseteq Y \subseteq Z\}$. Actually, $L$ are the most general patterns in $C$, and $R$ the most specific patterns in $C$. We can write $[L, R]$ for $C$. Observe that

**Proposition 1.** *Suppose $D^P$ and $D^N$ have no duplicate and no intersection and are of the same dimension. Then the set of emerging patterns from $D^P$ to $D^N$ is given by $[\{\{\}\}, D^P] - [\{\{\}\}, D^N]$.* □

Let $D^P = \{A_1, \ldots, A_n\}$ and $D^N = \{B_1, \ldots, B_m\}$. Then $[\{\{\}\}, D^P] - [\{\{\}\}, D^N] = [\{\{\}\}, \{A_1, \ldots, A_n\}] - [\{\{\}\}, \{B_1, \ldots, B_m\}] = [L, \{A_1, \ldots, A_n\}]$, where $L = min(\bigcup_i^n \{\{\, s_1, \ldots, s_m\} \mid s_j \in A_i - B_j, 1 \leq j \leq m\})$. This definition of $L$ is something we know how to optimize [11,12] using datamining ideas from Max-Miner's look-ahead [3], Apriori's candidate generation [1], and level-wise search [27].

## 4 Another Important Role of Databases: Laboratory Workflow

We have seen the role that database research has played in data integration and knowledge discovery in modern molecular biology research. We have also described two technologies that were very original when they were first developed. The first is Kleisli, introduced in 1994. It is the first broad-scale data integration system that employs the nested relational data model, an explicit data exchange format, and a mediator-wrapper architecture. These features greatly facilitated the incorporation of numerous biological data sources and applications into Kleisli. The second is PCL—and the idea of emerging patterns, introduced in 1998—which is a machine learning method quite distinct from traditional machine learning methods. It produces highly human-understandable rules and also achieves very good accuracy.

To wrap up this paper, we discuss a third area where database research has a key role to play—the workflow of laboratory information management systems (LIMS). LIMS are deployed in genomic laboratories right where the raw genomic data are generated. The informatics challenge there is to control laboratory workflow, while keeping track of the mountains of data being generated in a consistent and accountable fashion. A LIMS is thus a specialized database management system that focuses on the management of the two key aspects of a genome laboratory: Data and process.

Data have been a key achievement of the genome era—an era that has resulted in a multitude of new technologies that enable researchers to, say, sequence entire genomes [20] or scan tens of thousands of genes in a single chip experiment [15]. Such technological advancements have brought about challenging data management requirements for LIMS: (a) Voluminous data management. LIMS databases should be highly scalable as genome technologies have been designed to generate data *en masse*, and typical data entries in genome laboratories are often large capacity data items such as images and genetic sequences. (b) Heterogeneous data format handling. Laboratory data are often captured in the LIMS in a processed form (e.g., normalized peak heights instead of the entire electropherograms), requiring that the LIMS be able to access the proprietary file formats of different laboratory equipment to perform the necessary data extraction. A good LIMS should also capture useful data-associated user annotations, usually descriptive in nature and is best represented as free text. (c) Persistent and secure data archiving. The computerization of the laboratory and the emergence of bioinformatics have resulted in more and more laboratory data residing on computer systems in various digital formats. A file repository component must be included with a LIMS to persist, secure, and organize data files captured in the laboratory procedures. The recent FDA's requirement on "21 CFR Part 11" compliance has made it urgently necessary for drug companies to implement persistent and secure electronic archiving of laboratory data. A LIMS must therefore have built-in mechanisms to ensure the authenticity and integrity of the data.

Management of laboratory processes is the other focus of LIMS: (a) Sample tracking. Most laboratories implement a barcode system for tracking samples physically (e.g., blood, tissue, DNA). "Logical" tracking of laboratory samples, on the other hand, can be a complicated data management task in genome laboratories: A sample may be duplicated into multiple copies, extracted into various smaller fragments, loaded with others onto a single plate or array, reloaded with different groups of samples for another experimental procedures, and so on. LIMS databases must model such complex transformations of genomic objects. (b) Protocol tracking. Experimental protocols should be recorded in the LIMS and associated with the data generated, so that what has been done to a laboratory sample can be explained or redone. However, in an emerging field such as genomics, experimental protocols are continuously being re-engineered. LIMS databases must support an efficient versioning scheme for defining and updating protocols and tracking the changes made. (c) Inventory tracking. To maintain high throughput in laboratories, inventory tracking is an essential LIMS functionality so that the necessary laboratory materials (e.g., reagents, test-tubes, plates) are in stock and valid (for those with expiry dates). This means that all laboratory procedures must be modeled in fine details in the LIMS so that the database can automatically estimate current inventory based on usage.

Other LIMS-related operational database requirements include: (a) Instrument integration and interfacing. Integration is a key issue for LIMS, as it is not uncommon to employ multiple systems to conduct experiments in genome laboratories, and the LIMS is often expected to automate the operation of some of the laboratory processes in addition to mere data extraction from the different machines. (b) User and project management. A LIMS is often an enterprise solution in a genome institution, catering to a mixed community of users from laboratory technicians to research scientists and managers, each having his own use requirements in different experimental projects. The

LIMS must therefore satisfy the different user needs, and implement appropriate access control schemes for the different categories of users and projects.

Database research has enabled the transition of traditional biology into data-intensive genomics, by meeting the many challenging requirements in data generation, capture, and annotation through technological development such as Kleisli and operationally scalable implementation of LIMS. With the coming-of-age of bioinformatics in the genome era, computer analysis has become an integral part of genome knowledge discovery [34]. Post-genome LIMS not only impacts people within the wet laboratory but also extends into many other areas of the new biological discovery process, including bioinformaticists and biostatisticians in the "dry" labs. Traditional LIMS has focused on tracking data and processes in the physical environment of the wet laboratories; the post-genome database adventures in LIMS would be to also track the data and processes in the virtual environment of the dry labs, and relating them with those in the wet labs in one single consistent database system.

**Acknowledgement.** We thank Huiqing Liu for help in the TIS recognition.

## References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*, pp 487–499.
2. P. G. Baker and A. Brass. Recent development in biological sequence databases. *Curr. Op. Biotech.*, 9:54–58, 1998.
3. R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*, pp 85–93.
4. P. Buneman et al. Comprehension syntax. *SIGMOD Record*, 23:87–96, 1994.
5. P. Buneman et al. Principles of programming with complex objects and collection types. *TCS*, 149:3–48, 1995.
6. J. Chen et al. The Kleisli query system as a backbone for bioinformatics data integration and analysis. In *Bioinformatics: Managing Scientific Data*, Morgan Kaufmann. To appear.
7. T.M. Cover and P.E. Hart. Nearest neighbour pattern classification. *IEEE Trans. Info. Theory*, 13:21–27, 1967.
8. L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL'82*, pp 207–212.
9. S. Davidson et al. BioKleisli: A digital library for biomedical researchers. *Intl. J. Digit. Lib.*, 1:36–53, 1997.
10. Department of Energy. *DOE Informatics Summit Meeting Report*, 1993.
11. G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD'99*, pp 15–18.
12. J. Li et al. The space of jumping emerging patterns and its incremental maintenance algorithms In *ICML'00*, pp 551–558.
13. U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI'93*, pp 1022–1029,
14. D. Gerhold et al. DNA chips: promising toys have become powerful tools. *Trends Biochem. Sci.*, 24:168–173, 1999.
15. T.R. Golub et al. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.
16. L.M. Haas et al. DiscoveryLink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40:489–511, 2001.

17. A.G. Hatzigeorgiou. Translation initiation start prediction in human cDNAs with high accuracy. *Bioinformatics*, 18:343–350, 2002.
18. G. Jaeschke and H. J. Schek. Remarks on the algebra of non-first-normal-form relations. In *PODS'82*, pp 124–138.
19. M. Kozak. An analysis of 5'-noncoding sequences from 699 vertebrate messenger RNAs. *NAR*, 15:8125–8148, 1987.
20. E.S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409:861–921, 2001.
21. P. Langley et al. An analysis of Bayesian classifier. In *AAAI'92*, pp 223–228.
22. J. Li et al. Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. *Bioinformatics*, 2002. To appear.
23. J. Li and L. Wong. Geography of differences between two classes of data. In *PKDD'02*, pp 325–337.
24. L. Libkin and L. Wong. Query languages for bags and aggregate functions. *JCSS*, 55(2):241–272, October 1997.
25. H. Liu and R. Sentiono. Chi2: Feature selection and discretization of numeric attributes. In *Proc. IEEE 7th Intl. Conf. on Tools with Artificial Intelligence*, pp 338–391, 1995.
26. A. Makinouchi. A consideration on normal form of not necessarily normalised relation in the relational data model. In *VLDB'77*, pp 447–453.
27. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1:241–258, 1997.
28. Y. Papakonstantinou et al. Object exchange across heterogenous information sources. In *ICDE'95*, pp 251–260.
29. P. Pearson et al. The GDB human genome data base anno 1992. *NAR*, 20:2201–2206, 1992.
30. A.G. Pedersen and H. Nielsen. Neural network prediction of translation initiation sites in eukaryotes: Perspectives for EST and genome analysis. *ISMB*, 5:226–233, 1997.
31. J.R. Quinlan. *C4.5: Program for Machine Learning*. Morgan Kaufmann, 1993.
32. D. E. Rumelhart et al. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
33. G. D. Schuler et al. Entrez: Molecular biology database and retrieval system. *Methods Enzymol.*, 266:141–162, 1996.
34. D.B. Searls. Using bioinformatics in gene and drug discovery. *DDT*, 5:135–143, 2000.
35. S.J. Thomas and P.C. Fischer. Nested relational structures. In *Advances in Computing Research: The Theory of Databases*, pp 269–307, 1986.
36. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
37. P. Wadler. Comprehending monads. *Math. Struct. Comp. Sci.*, 2:461–493, 1992.
38. L. Wong. Normal forms and conservative extension properties for query languages over collection types. *JCSS*, 52:495–505, 1996.
39. L. Wong. Kleisli, a functional query system. *JFP*, 10:19–56, 2000.
40. L. Wong. Kleisli, its exchange format, supporting tools, and an application in protein interaction extraction. In *BIBE'00*, pp 21–28.
41. E.J. Yeoh et al. Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer Cell*, 1:133–143, 2002.
42. F. Zeng et al. Using feature generation and feature selection for accurate prediction of translation initiation sites. In *GIW'02*. To appear.
43. A. Zien et al. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16:799–807, 2000.

# Incremental Validation of XML Documents

Yannis Papakonstantinou and Victor Vianu*

Computer Science and Engineering, University of California at San Diego
{yannis, victor}@cs.ucsd.edu

**Abstract.** We investigate the incremental validation of XML documents with respect to DTDs and XML Schemas, under updates consisting of element tag renamings, insertions and deletions. DTDs are modeled as extended context-free grammars and XML Schemas are abstracted as "specialized DTDs", allowing to decouple element types from element tags. For DTDs, we exhibit an $O(m \log n)$ incremental validation algorithm using an auxiliary structure of size $O(n)$, where $n$ is the size of the document and $m$ the number of updates. For specialized DTDs, we provide an $O(m \log^2 n)$ incremental algorithm, again using an auxiliary structure of size $O(n)$. This is a significant improvement over brute-force re-validation from scratch.

## 1 Introduction

The emergence of XML as a standard representation format for data on the Web has led to a proliferation of databases that store, query, and update XML data. Typically, valid XML documents must conform to a specified type that places structural constraints on the document. When an XML document is updated, it has to be verified that the new document still satisfies its type. Doing this efficiently is a challenging problem that is central to many applications. Brute-force validation from scratch is often not practical, because it requires reading and validating the entire database following each update. Instead, it is desirable to develop algorithms for incremental validation. However, this approach has been largely unexplored. In this paper we investigate the efficient incremental validation of updates to XML documents.

An XML document can be viewed abstractly as a tree of nested elements. The basic mechanism for specifying the type of XML documents is provided by Document Type Definitions (DTDs) [W3C98]. DTDs can be abstracted as extended context-free grammars (CFGs). Unlike usual CFGs, the productions of extended CFGs have regular expressions on their right-hand sides. An XML document satisfies a DTD if its abstraction as a tree is a derivation tree of the extended CFG corresponding to the DTD. A more recent XML typing mechanisms uses the XML Schema standard [W3C01], which extends DTDs in several ways. Most notable is the ability to decouple the type of an element from its label. In this paper we abstract XML schemas by *specialized* DTDs, that capture

* Authors supported in part by the NSF under grant numbers IRI-9734548, IRI-9221268 and Digital Government 9983510.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 47–63, 2003.

precisely this aspect. It is a well-known and useful fact that specialized DTDs define precisely the regular languages of unranked trees, and so are equivalent to top-down (and bottom-up) non-deterministic tree automata.

Verifying that a word satisfies a regular expression[1] is the starting point in checking that an XML document satisfies a DTD. An obvious way to do this following an update is to verify it from scratch, i.e. run the updated sequence of labels through the non-deterministic finite automaton (NFA) corresponding to the regular expression. However, this requires $O(n)$ steps, under any reasonable set of unit operations, where $n$ is the length of the sequence (note that, in complexity-theoretic terms, membership of a word in a regular language is complete in $NC^1$ under DLOGTIME reductions [Vol99].) We can do better by using incremental validation, relying on an appropriate auxiliary data structure. Indeed, we provide such a structure and corresponding incremental validation algorithm that, given a regular expression $r$, a string $s$ of length $n$ that satisfies $r$, and a sequence of $m$ updates (inserts, deletes, label renamings) on $s$, checks[2] in $O(m \log n)$ whether the updated string satisfies $r$. The auxiliary structure we use materializes in advance relations that describe state transitions resulting from traversing certain substrings in $s$. These are placed in a balanced tree structure that is maintained similarly to B-trees and is well-behaved under insertions and deletions. The size of the auxiliary structure is $O(n)$. In addition, we provide an $O(m \log n)$ time algorithm that maintains the auxiliary structure, so that subsequent updates can also be incrementally validated.

Our approach to incremental validation of trees with respect to specialized DTDs builds upon the incremental validation algorithm for strings. DTDs turn out to be easier to validate than specialized DTDs. Indeed, based on the algorithm for string validation, incremental validation of $m$ updates to a tree $T$ with respect to a DTD can be done in time $O(m \log |T|)$ using an auxiliary structure of size $O(|T|)$ which can also be maintained in time $O(m \log |T|)$. Specialization introduces another degree of complexity. Intuitively, this is due to the fact that an update to a single node may have global repercussions for the typing of the tree. This stands in contrast with DTDs without specialization, where a single update to a node needs to be validated only with respect to the type of its parent and the sequence of its children, so has local impact on type checking.

A straightforward extension of the incremental validation for DTDs yields an algorithm of time complexity $O(m\ \mathit{depth}(T) \log |T|)$ using an auxiliary structure of size $O(|T|)$. However, this is not satisfactory when the tree is narrow and deep. In the worst case, $\mathit{depth}(T) = |T|$. To overcome this, we develop a more subtle approach that has the following main ingredients: First, the unranked tree $T$ representing the XML document is mapped into a binary tree encoding that allows us to unify the horizontal and vertical components of validation. Then the specialized DTD is translated into a bottom-up non-deterministic tree automaton accepting precisely the encodings of valid documents. Finally, an in-

[1] A word *satisfies* a regular expression if it belongs to the corresponding language.

[2] For readability, we provide here the complexity with respect to the string and update sequence, for fixed (specialized) DTD or regular expression. The combined complexity is spelled out in the paper.

cremental validation algorithm for binary trees with respect to tree automata is developed, based on a divide-and-conquer strategy that focuses on computations along certain paths in the tree chosen to appropriately divide the work. Auxiliary structures are associated to each of these paths. The resulting incremental validation algorithm has time complexity $O(m \log^2 |T|)$ and uses an auxiliary structure of size $O(|T|)$.

*Related Work.* Closely related to incremental validation is incremental parsing, which is key to incremental program compilation. Research on incremental parsing has focused on LR parsing [GM80,WG98,JG82,Lar95,Pet95] and LL (recursive descent parsing) [MPS90,Li95,Lin93], since programming languages are typically described by LR(0), LR(1), LL(1), LALR(1) and LL(1) grammars. All techniques start by parsing the input text and produce a parse tree, which is typically annotated with auxiliary information. The parse tree is updated as a result of the updates to the input text. A typical theme of the incremental parsing techniques is identifying minimal structural units of the parse tree that are affected by the modifications (see [GM80] for LR(0) parsing and [Lar95] for a generalization to LR($k$).) However, the performance of the incremental parsing algorithms is hard to compare to our validation algorithm because of the differences in settings and goals, which typically involve minimization of the changes on the parse tree. Indeed, the best-case performance of incremental parsers will generally beat the one of our regular expression validation algorithm, which always takes $O(\log n)$ steps for a single update. This is because incremental parsers take advantage of natural "termination points" used in programming languages syntax [Lin93], that typically occur close to the update. Logarithmic complexity in the size of the string is achieved for LALR grammars by [WG98] but only if the grammar is such that its parse trees have depth $O(\log n)$ for a string of length $n$. One can easily see that there are LALR grammars that do not meet this property, and neither do the CFGs corresponding to DTDs. Furthermore, [WG98] require that the interpretation of iterative sequences be independent of the context.

The complexity of validation is related to that of membership of a word in a regular language, and of a tree in a regular tree language. The problem of word membership in a regular language is known to be complete in uniform $NC^1$ under DLOGTIME reductions [Loh01] and acceptance of a tree over a ranked alphabet by a tree automaton is complete in uniform $NC^1$ under DLOGTIME reductions if the tree is presented in prefix notation [Vol99], and complete in LOGSPACE if the tree is presented as a list of its edges [Seg02]. To our knowledge, no complexity results exist on the incremental variants of these problems, with the exception of a result of [PI97] discussed below.

Incremental evaluation of queries by first-order means is studied by [DS95] using the notion of first-order incremental evaluation systems (FOIES) A related descriptive complexity approach to incremental computation is developed by Patnaik and Immerman in [PI97]. They define the dynamic complexity class Dyn-FO (equivalent to FOIES), consisting of properties that can be incrementally verified by first-order means. They exhibit various problems in Dyn-FO,

such as multiplication, graph connectivity, and bipartiteness. Most relevant to our work, they show that membership of a word in a regular language is in Dyn-FO. For label renamings, they sketch an approach similar to ours. The incremental algorithm and auxiliary structure for node insertions and deletions that modify the length of the string are not spelled out. Also, no extension to regular tree languages is discussed. The study in [PI97] is pursued in [HI02], where an extension of Dyn-FO is introduced and it is shown that the single-step version of the circuit value problem is complete in Dyn-FO under certain reductions. Complexity models of incremental computation are considered in [MSVT94]. The focus is on the classes *incr*-POLYLOGTIME (*incr*-POLYLOGSPACE) of properties that can be incrementally verified in polylogarithmic time (space). Interesting connections to parallel complexity classes are exhibited, as well as complete problems for classical complexity classes under reductions in the above incremental complexity classes.

*Organization.* Section 2 presents our abstraction of XML documents, DTDs, and XML Schemas. We also spell out formally the incremental validation problem and the assumptions made in our complexity analysis. In Section 3 we examine the incremental validation of strings with respect to regular expressions and develop the core divide-and-conquer strategy used later for DTD validation. Section 4 presents the validation algorithm for DTDs. Finally, Section 5 presents the full algorithm for specialized DTDs yielding $O(m \log^2 |T|)$ incremental validation. Section 6 contains some concluding remarks and future work.

## 2 Basic Framework

We introduce here the basic formalism used throughout the paper, including our abstractions of XML documents, DTDs, and XML Schemas. We also recall basic definitions relating to tree automata.

*Labeled ordered trees.* We abstract XML documents as labeled ordered trees. Our abstraction ignores data values present in XML documents, because their validation with respect to an XML Schema is trivial.

An *ordered labeled tree* over finite alphabet $\Sigma$ is a pair $T = \langle t, \lambda \rangle$, where $t$ is an ordered tree and $\lambda$ is a mapping associating to each node $n$ of $t$ a label $\lambda(n) \in \Sigma$. Trees are assumed by default to be unranked, i.e. there is no fixed bound on the number of children each node may have. The set of all labeled ordered trees over $\Sigma$ is denoted by $\mathcal{T}_\Sigma$. We sometimes denote a tree consisting of a root $v$ with subtrees $T_1 \dots T_k$ by $v(T_1 \dots T_k)$. We will also consider binary trees, where each node has at most two children. If every internal node has *exactly* two children, the binary tree is called *complete*.

We assume a representation of trees that allows one to find in $O(1)$ (i) the label, (ii) the parent, (iii) the immediate left (right) sibling, and (iv) the first child of a specified node.

*Types and DTDs.* As usual, we define XML document types in terms of the document's structure alone, ignoring data values. The basic specification method is (an abstraction of) DTDs. A DTD consists of an extended context-free grammar over alphabet $\Sigma$ (we make no distinction between terminal and non-terminal symbols). In an extended CFG, the right-hand sides of productions are regular expressions over $\Sigma$. An ordered labeled tree $\langle t, \lambda \rangle$ over $\Sigma$ satisfies a DTD $d$ if the tree $\langle t, \lambda \rangle$ is a derivation tree of the grammar.

The start symbol of a DTD $d$ is denoted by $root(d)$. We can assume without loss of generality that for each $a \in \Sigma$ the DTD has a single rule $a \ \rightarrow r_a$ with $a$ on the left-hand side. and we denote by $N_a$ a standard non-deterministic finite-state automaton (NFA) recognizing the language $r_a$. The set of labeled trees satisfying a DTD $d$ is denoted by $sat(d)$.

An NFA is a 5-tuple $N = \langle \Sigma, Q, q_0, F, \delta \rangle$ where $\Sigma$ is a finite alphabet, $Q$ is a finite set of *states*, $q_0 \in Q$ is the *start state*, $F \subseteq Q$ is the set of *final states*, and $\delta$ is a mapping from $\Sigma \times Q$ to $\mathcal{P}(Q)$. A string $a_1 \ldots a_n$ is accepted by $N$ iff there exists a mapping $\sigma : \{1, \ldots, n\} \rightarrow Q$ such that $\sigma(a_1) \in \delta(a_1, q_0)$, $\sigma(a_n) \in F$, and for each $i, 1 \leq i < n$, $\sigma(a_{i+1}) \in \delta(a_{i+1}, \sigma(a_i))$. The set of strings accepted by $N$ is denoted $L(N)$. $N$ is a *deterministic finite-state automaton* (DFA) iff $\delta$ returns singletons on each input. Recall that for each regular expression $r$ there exists an NFA $N$ whose number of states is linear in $r$, such that $N$ accepts the regular language $r$. In general, a DFA accepting $r$ requires exponentially many states wrt $r$. However, for certain classes of regular expressions, the corresponding DFA remains linear in the expression. One such class consists of the 1-unambiguous regular languages [BKW98]. This is relevant in the context of XML types, since DTDs and XML Schemas require the regular expressions used to specify the contents of elements to be 1-unambiguous.

An important limitation of DTDs is the inability to separate the *type* of an element from its *name*. We abstract this mechanism using the notion of *specialized DTD* (studied in [PV00] and equivalent to formalisms proposed in [BM99,CDSS98]).

**Definition 1.** *Specialized DTD* A specialized DTD *is a 4-tuple* $\langle \Sigma, \Sigma^t, d, \mu \rangle$ *where* $\Sigma$ *is a finite alphabet of labels,* $\Sigma^t$ *is a finite alphabet of* types*,* $d$ *is a DTD over* $\Sigma^t$ *and* $\mu$ *is a mapping from* $\Sigma^t$ *to* $\Sigma$*.*

Intuitively, $\Sigma^t$ provides, for each $a \in \Sigma$, a set of types associated to $a$, namely those $a^t \in \Sigma^t$ for which $\mu(a^t) = a$. For example, assume that we want to describe car ads. Ad elements belong either to an element UC, which has all used car ads, or an element NC, which has all new car ads. The ads belonging to used cars have both model and year, while the new car ads have only model. The DTD listed on the left side below cannot capture this point. The specialized DTD on the right solves the problem by using two types for the element *ad*: a type $ad^n$ whose content is just a "model" type, and a type $ad^u$ whose content is "model" and "year".

| | |
|---|---|
| $root : dealer$ | $root : d^t$ |
| $dealer \rightarrow UC\ NC$ | $d^t \rightarrow UC^t\ NC^t \quad \mu(d^t) = dealer$ |
| $UC \rightarrow ad^*$ | $UC^t \rightarrow (ad^u)^* \quad \mu(UC^t) = UC$ |
| $NC \rightarrow ad^*$ | $NC^t \rightarrow (ad^n)^* \quad \mu(NC^t) = NC$ |
| $ad \rightarrow model\ (year \mid \epsilon)$ | $ad^u \rightarrow m^t\ y^t \quad \mu(ad^u) = ad$ |
| $model \rightarrow \epsilon$ | $ad^n \rightarrow m^t \quad \mu(ad^n) = ad$ |
| $year \rightarrow \epsilon$ | $m^t \rightarrow \epsilon \quad \mu(m^t) = model$ |
| | $y^t \rightarrow \epsilon \quad \mu(y^t) = year$ |

Note that $\mu$ induces a homomorphism on words over $\Sigma^t$, and also on trees over $\Sigma^t$ (yielding trees over $\Sigma$). We also denote by $\mu$ the induced homomorphisms.

Let $\tau = \langle \Sigma, \Sigma^t, d, \mu \rangle$ be a specialized DTD. A tree $t$ over $\Sigma$ satisfies $\tau$ (or is *valid* wrt $\tau$) if $t \in \mu(sat(d))$. Thus, $t$ is a homomorphic image under $\mu$ of a derivation tree in $d$. Equivalently, a labeled tree over $\Sigma$ is valid if it can be "specialized" to a tree that is valid with respect to the DTD over the alphabet of types. The set of all trees over $\Sigma$ that are valid w.r.t. $\tau$ is denoted $sat(\tau)$. When $\tau$ is clear from the context, we simply say that a tree is *valid*.

*Tree automata.* There is a powerful connection between specialized DTDs and *tree automata*: they are precisely equivalent, and define the *regular tree languages* [BKMW01]. We will make use of this connection in the paper.

Tree automata are devices whose purpose is to accept or reject an input tree. Classical tree automata are defined on complete binary trees. As in the case of string automata, there are several equivalent variants. We use bottom-up non-deterministic tree automata on complete binary trees. (For technical reasons that will become clear shortly, we assume that all leaves have the same label #.)

**Definition 2.** *A bottom-up non-deterministic tree automaton (BNTA) is a 5-tuple $A = \langle \Sigma, Q, Q_0, q_f, \delta \rangle$ where $\Sigma$ is a finite alphabet, $Q$ is a finite set of* states*, $Q_0$ is the set*[3] *of* start states *($Q_0 \subseteq Q$), $q_f$ is the* accept state *($q_f \in Q$) and $\delta$ is a mapping from $\Sigma \times Q \times Q$ to $\mathcal{P}(Q)$.*

*A tree $T = \langle t, \lambda, \rangle$ is* accepted *by the automaton $A$ iff there is a mapping $\sigma$ from the nodes of $t$ to $Q$ such that: (i) if $n$ is a leaf then $\sigma(n) \in Q_0$, (ii) if $n$ is an internal node with children $n_1, n_2$ then $\sigma(n) \in \delta(\lambda(n), \sigma(n_1), \sigma(n_2))$, and (iii) if $n$ is the root then $\sigma(n) = q_f$. The set of trees accepted by $A$ is denoted by $\mathcal{T}(A)$.*

There is a *prima facie* mismatch between DTDs and tree automata: DTDs describe unranked trees, whereas classical automata describe binary trees. We resolve the mismatch by encoding unranked labeled ordered trees as binary trees, which also enable the key divide-and-conquer algorithms of Section 5.

*The incremental validation problem.* Given a (specialized) DTD $\tau$, a tree $T \in sat(\tau)$, and a sequence $s$ of updates to $T$ yielding another tree $T'$, we wish to efficiently check if $T' \in sat(\tau)$. In particular, the cost should be less than re-validation of $T'$ from scratch. The individual updates are the following:

[3] Some definitions of BNTA require a single start state for each leaf symbol, and allow a set of final states. Having multiple start states and a single final state is a harmless variation, convenient here for technical reasons.

(a) replace the current label of a specified node by another label,
(b) insert a new leaf node after a specified node,
(c) insert a new leaf node as the first child of a specified node, and
(d) delete a specified leaf node.

We allow some cost-free one-time pre-processing to initialize incremental validation, such as computing the NFA corresponding to the regular expressions used by the DTDs. We will also initialize and then maintain an auxiliary structure $\mathcal{A}(T)$ to help in the validation. The cost of the incremental validation algorithm is evaluated with respect to:
(a) the time to validate $T'$ using $T$ and $\mathcal{A}(T)$, as a function of $|T|$ and $|s|$
(b) the time needed to compute $\mathcal{A}(T')$ from $T, s$, and $\mathcal{A}(T)$,
(c) the size of the auxiliary structure $\mathcal{A}(T)$ as a function of $|T|$.

The analysis will also make explicit the combined complexity taking into account the specialized DTD.

## 3 Incremental Validation of Sequences

We consider the incremental validation of strings with respect to a regular language specified by an NFA. Consider an NFA $N = \langle \Sigma, Q, q_0, F, \delta \rangle$ and a sequence $v_1 \dots v_n$ of nodes with labels $\lambda(v_i) = a_i, i = 1, \dots, n$ such that $a_1 \dots a_n \in L(N)$. When there is no confusion we blur the distinction between the sequence of elements and the string of labels.

Consider a sequence of element renamings $u(v_{i_1}, b_1), \dots, u(v_{i_m}, b_m)$, where $1 \leq i_1 < i_2 < \dots < i_m \leq n$. The renaming $u(v_{i_j}, b_j)$ requires that the label of element $v_{i_j}$ be renamed to $b_j$. We would like to efficiently check whether the updated string $a_1 \dots a_{i_1-1} b_1 a_{i_1+1} \dots a_{i_m-1} b_m a_{i_m+1} \dots a_n \in L(N)$. Validating the new string from scratch by running it through $N$ takes $O(n|Q|^2 \log |Q|)$. In a similar fashion, validating $m$ insertions from scratch is $O((n+m)|Q|^2 \log |Q|)$. We can do better in all cases by maintaining the following auxiliary information.

The *transition relation* $T_{[v_i \dots v_j]}$ of a sequence $v_i \dots v_j$ is $\{\langle p, q \rangle \mid p, q \in Q, q \in \delta(p, \lambda(v_i) \dots \lambda(v_j))\}$. Note that $T_{[v_i \dots v_j]} = T_{[v_i \dots v_k]} \circ T_{[v_{k+1} \dots v_j]}$ where $i \leq k < j$ and $\circ$ denotes composition of binary relations, which can be computed in $O(|Q|^2 \log |Q|^2) = O(|Q|^2 \log |Q|)$ using sort-merge join (each relation is sorted in $O(|Q|^2 \log |Q|)$ and then they are merged in $O(|Q|^2)$).

We use as auxiliary structure a set of transition relations organized into a *transition relation tree* $\mathcal{T}$ that allows validating a sequence of $m$ updates to a string of length $n$ in $O(m|Q|^2 \log |Q| \log n)$ time, and can be maintained in the same. The size of the tree is $O(|Q|^2 n)$. Note that the approach below is similar to that briefly sketched in [PI97] for label renamings and extends [PI97] to capture insertions and deletions by maintaining a balanced transition relation tree.

The particular balanced tree variant we use is the 2-3 tree, which was a precursor to B-trees [CLR]. Each node contains 3 cells. Each cell is either empty or contains a transition relation $T_s$ corresponding to some subsequence $s$ of $v_1 \dots v_n$, conforming to the rules described below. At most one of the 3 cells in

a node can be empty, assuming $n \geq 2$. Each nonempty cell is either at a leaf or has one node (with three cells) as a child. The following rules apply to the transition relations stored in the cells:

- if the root has two nonempty cells containing the relations $T_{s_1}$ and $T_{s_2}$ (resp. three cells containing the relations $T_{s_1}, T_{s_2}$ and $T_{s_3}$) then $T_{s_1} \circ T_{s_2} = T_{[v_1 \ldots v_n]}$ (resp. $T_{s_1} \circ T_{s_2} \circ T_{s_3} = T_{[v_1 \ldots v_n]}$);
- if an internal cell contains a relation $T_s$ and its child node contains $T_{s_1}$, $T_{s_2}$ (resp. $T_{s_1}$, $T_{s_2}$, and $T_{s_3}$) then $T_s = T_{s_1} \circ T_{s_2}$ (resp. $T_s = T_{s_1} \circ T_{s_2} \circ T_{s_3}$);
- the sequence of non-empty leaf cells is $T_{s_1} \ldots T_{s_n}$ and $T_{s_i} = T_{[v_i]}$, $1 \leq i \leq n$.

We also maintain pointers providing in $O(1)$, for each element $v_i$ in the input sequence, the leaf cell $T_{[v_i]}$. Note that the position of the element is never recorded explicitly and, similarly, we never compute explicitly the subsequence associated to each $T_s$. The left part of Figure 1 shows a sequence of seven nodes, several subsequences, and the corresponding transition relation tree. Note that the subscript of a node does not necessarily indicate its position in the string. Each sequence $s_i$ is the singleton sequence $n_i$, for $i \in \{1, 2, 3, 5, 6, 7, 9\}$.

The requirement of having 3 cells per node of which at least 2 are nonempty ensures that the tree remains balanced and of depth $O(\log n)$ as it is updated. This follows from the standard analysis of B-tree behavior under the maintenance algorithm [GMUW01]. In a disk-based implementation one should set the maximum number of cells per node to the number of items that fit in one disk page. At any rate, the depth of the tree is $O(\log n)$ and its size is $O(n|Q|^2)$.

*Validation and Maintenance.* Consider a sequence of renamings $u(v_{i_1}, b_1), \ldots, u(v_{i_m}, b_m)$, where $i_1 < i_2 < \ldots < i_m$. The updated string of element labels is $\lambda(v_1) \ldots \lambda(v_{i_1-1}) b_1 \lambda(v_{i_1+1}) \ldots \lambda(v_{i_m-1}) b_m \lambda(v_{i_m+1}) \ldots \lambda(v_n)$. Note that the relations $T_s$ affected by the updates are those laying on the path from a leaf $T_{[v_{i_j}]}$ $(1 \leq j \leq m)$ to the root of the transition relation tree $\mathcal{T}$. Let $\mathcal{I}$ be the set of such relations, and note that its size is at most $m \log n$.

The tree $\mathcal{T}$ can now be updated by recomputing the $T_s$'s in $\mathcal{I}$ bottom-up as follows: First, the leaves $T_{[v_{i_j}]} \in \mathcal{I}$ are set to $\{\langle p, q\rangle \mid q \in \delta(p, b_j)\}$, $1 \leq j \leq m$. Then for every internal cell that contains a $T_s \in \mathcal{I}$ and its child node has two nonempty cells with $T_{s_1}$ and $T_{s_2}$ (resp. three cells with $T_{s_1}$, $T_{s_2}$ and $T_{s_3}$) for which at least one has been recomputed is replaced by $T_{s_1} \circ T_{s_2}$ (resp. $T_{s_1} \circ T_{s_2} \circ T_{s_3}$). Thus, at most $m \log n$ $T_s$'s have been recomputed, each in time $O(|Q|^2 \log |Q|)$, yielding a total time of $O(m|Q|^2 \log |Q| \log n)$.

The validation of the string is now trivial: it is enough to check, in the updated auxiliary structure, that $\langle q_0, f\rangle \in T_{[v_1, \ldots, v_n]}$ for some $f \in F$. $T_{[v_1, \ldots, v_n]}$ is the composition of the sets $T_s$ in the cells of the root node. Thus, validation is also done in time $O(m|Q|^2 \log |Q| \log n)$.

If the update is the insertion or deletion of new elements, the maintenance algorithm mimics the one for B-trees. In particular, recall that if nodes in a B-tree become too full as the result of an insertion they are split, and if they contain fewer than two non-empty cells as a result of a deletion they are either

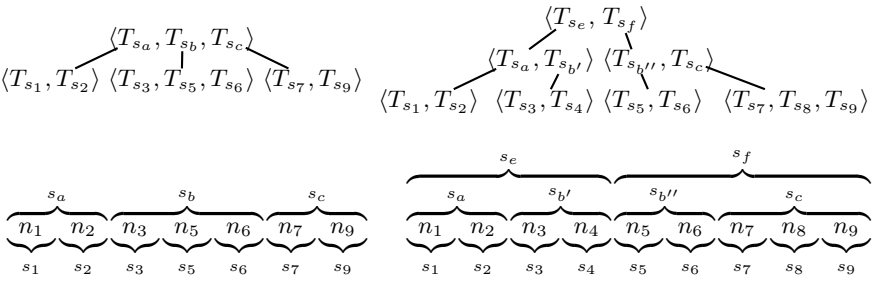



**Fig. 1.** A $\mathcal{T}$ tree before and after the insertion of nodes $n_4$ and $n_8$

merged with a sibling node or non-empty cells are transferred from a sibling node. The node splits and merges may propagate all the way to the root. Due to the similarity to classical B-tree maintenance we omit the details and instead illustrate the insertion algorithm with an example. If a new node $n_8$ is inserted in the left string of Figure 1 after $n_7$, we insert $T_{s_8}$ in the node $\langle T_{s_7}, T_{s_9}\rangle$, as shown in the right side of Figure 1, and we revise $T_{s_c}$, which becomes $T_{s_7} \circ T_{s_8} \circ T_{s_9}$. Then, the insertion of a node $n_4$ following $n_3$ leads to splitting the node $\langle T_{s_3}, T_{s_5}, T_{s_6}\rangle$ into $\langle T_{s_3}, T_{s_4}\rangle$ and $\langle T_{s_5}, T_{s_6}\rangle$. The relation $T_{s_b}$ is deleted and two new relations $T_{s_{b'}}$ and $T_{s_{b''}}$ are inserted into $\langle T_{s_a}, T_{s_b}, T_{s_c}\rangle$, which leads to a new split and a new root. The result tree is shown in the right side of Figure 1.

In the worst case, when an insertion in a leaf node results in splits propagating all the way to the root, we need to recompute $2 \log n$ new relations (one at the leaf level, one at the new root, and $2(\log n - 1)$ at the internal nodes). Hence, the worst case complexity is $O(|Q|^2 \log |Q| \log n)$. Deletion proceeds similarly and may lead to node merging or root deletion, with the same complexity. As in the case of B-trees, the maintenance algorithm guarantees that $\mathcal{T}$ always has depth $O(\log n)$ for strings of length $n$. Altogether, maintenance of $\mathcal{T}$ after $m$ updates takes time $O(m|Q|^2 \log |Q| \log n)$.

*1-unambiguous regular expressions.* As discussed earlier, XML Schemas require regular expressions used in type definitions to be 1-unambiguous. If $r$ is a 1-unambiguous regular expression, the corresponding DFA is of size linear in $r$. In this case, the relations $T_s$ used in the above auxiliary structure have size $O(|Q|)$ rather than $O(|Q|^2)$. This reduces the size of the auxiliary structure to $O(|Q|n)$ and the complexity of maintenance and validation to $O(m|Q| \log |Q| \log n)$.

## 4 Incremental DTD Validation

The incremental validation of DTDs extends the divide-and-conquer algorithm for incremental validation of strings described in Section 3. Let $d$ be a DTD, $T = \langle t, \lambda\rangle$ a labeled tree satisfying $d$, and consider first updates consisting of a sequence of $m$ label modifications yielding a new tree $T' = \langle t', \lambda'\rangle$. To check that $T'$ satisfies $d$, we must verify that for each node $v$ in $t'$ with children $v_1 \dots v_n$ for which at least one label was modified, the sequence of labels $\lambda'(v_1) \dots \lambda'(v_n)$

belongs to $r_{\lambda'(v)}$. If the label of $v$ has not been modified, i.e. $\lambda(v) = \lambda'(v)$, then validation can be done using the divide-and-conquer algorithm described in Section 3 for strings. However, if the label of $v$ has been modified, so that $\lambda(v) \neq \lambda'(v)$, the sequence $\lambda'(v_1) \dots \lambda'(v_n)$ has to be validated with respect to the new regular language $r_{\lambda'(v)}$ rather than $r_{\lambda(v)}$. Thus, it would seem that, in this case, validation has to start again from scratch. To avoid this, we preemptively maintain information about the validity of each string of siblings with respect to *all* regular languages $r_a$ for $a \in \Sigma$. To this end we maintain some additional auxiliary information. Specifically, for each sequence of siblings in the tree, we compute the transitions relations $T_s$ of the divide-and-conquer algorithm described in Section 3, for each NFA $N_a$ corresponding to $r_a$, and $a \in \Sigma$. We denote the sets $T_s$ for a particular $a \in \Sigma$ by $T_s^a$. Since the auxiliary structure for each fixed NFA and string of length $n$ has size $O(|Q|^2 n)$ (where $Q$ is the set of states of the NFA), the size of the new auxiliary structure is at most $O(|\Sigma||d|^2|T|)$, where $|T|$ is the size of $T$ and $|d| = max\{|r_a| \mid a \to r_a \in d\}$. The maintenance of the auxiliary structure is done in the same way as in the string case, at a cost of $O(m|\Sigma||d|^2 \log |d| \log |T|)$ for a sequence of $m$ modifications. Finally, the updated tree $T'$ is valid wrt $d$ if for each node $v$ with label $a$ in $T'$ such that either $v$ or one of its children has been updated, $\langle q_0, f \rangle$ is in the relation $T_s^a$ where $s$ is the list of children of $v$, $q_0$ is the start state of $N_a$, and $f$ is one of its final states. Each such test takes $O(|d|^2 \log |d|)$ and the number of tests is $m$ in the worst case. This yields a total validation time of $O(m|\Sigma||d|^2 \log |d| \log |T|)$.

Insertions and deletions of leaves are handled by a straightforward extension of the B-tree approach outlined in Section 3.

*Specialized DTDs: a first attempt.* Specialized DTDs add another degree of complexity to the update validation problem, by allowing to associate different types with a given element label. Thus, an update to a single node may have global repercussions for the typing of the tree. The above validation algorithm for DTDs can be extended to take this into account (details omitted). The complexity of the resulting validation algorithm is $O(m|\Sigma^t||d|^2 \log |d| \mathit{depth}(T) \log |T|)$. Note that for fixed specialized DTD and update sequence, this is $O(\mathit{depth}(T) \log |T|)$. Thus, the algorithm works well for shallow trees. However, in the worst case $\mathit{depth}(T)$ could equal $|T|$, in which case the complexity is $O(|T| \log |T|)$. This is not satisfactory. In the next section, we develop a strategy that reduces the overall maintenance and validation cost to $O(\log^2 |T|)$.

## 5 Incremental Validation via Binary Trees Encodings

In this section we develop an incremental validation algorithm for specialized DTDs of complexity $O(\log^2 |T|)$ for fixed DTD and update sequence. To this end we develop a divide-and-conquer strategy that unifies the horizontal and vertical components of validation. We make use of a representation of unranked trees as complete binary trees and reduce the problem of validating specialized DTDs on unranked trees to that of acceptance of the binary tree encodings by a corresponding bottom-up tree automaton.

*Binary tree encoding of unranked trees.* We next describe the encoding of unranked trees as binary trees. We use one of the standard encodings in the literature (e.g, see [Nev02]). To each unranked labeled ordered tree $T = \langle t, \lambda \rangle$ over alphabet $\Sigma$ we associate a binary tree $enc(T)$ over alphabet $\Sigma_{\#} = \Sigma \cup \{\#\}$, where $\# \notin \Sigma$. The input of *enc* is a (possibly empty) sequence of unranked trees over $\Sigma$, and the output is a complete binary tree over $\Sigma_{\#}$. The mapping *enc* is defined recursively as follows (where $\bar{T}_0$ and $\bar{T}$ are sequences of trees, possibly $\epsilon$, and $n_0$ is a single node):

- $enc(\epsilon) = \#$
- $enc(n_0(\bar{T}_0)\ \bar{T}) = n_0(enc(\bar{T}_0), enc(\bar{T}))$

For example, a tree $T$ and its encoding $enc(T)$ are shown in Figure 2 (neglect for now the boxes and bold letters).

We would like to reduce the validation of unranked trees $T$ wrt a specialized DTD $\tau$ to the question of whether $enc(T)$ is accepted by a bottom-up nondeterministic tree automaton. To this end, we show the following result (a variant of known results on equivalences of specialized DTDs and tree automata, see [Nev02]):

**Lemma 1.** *For each specialized DTD $\tau = \langle \Sigma, \Sigma^t, d, \mu \rangle$ there exists a BNTA $A_\tau$ over $\Sigma_{\#}$ whose number of states is $O(|\Sigma^t||d|)$, such that $\mathcal{T}(A_\tau) = \{enc(T) \mid T \in sat(\tau)\}$.*

Our approach is based on reducing the validation of unranked trees with respect to specialized DTDs to the validation of their binary encodings with respect to the corresponding BNTA, say $A = \langle \Sigma, Q, Q_0, q_f, \delta \rangle$. As before, the problem really amounts to efficiently updating the auxiliary structure associated with the input. In our case, the auxiliary structure will include (among other information to be specified shortly) the binary encoding $enc(T)$ of the input $T$, and will provide, for each node $v$ in $enc(T)$, the set $types(v)$ consisting of the possible states of $A$ at node $v$ after consuming the subtree rooted at $v$. Once the auxiliary structure is updated, validity amounts to checking that $types(root(enc(T)))$ contains the accept state of $A$, where $T$ is the updated tree. The strategy for updating the types associated with nodes applies the divide-and conquer strategy for string validation to certain paths in the tree, chosen to appropriately divide the work. More precisely, we will select, in every subtree $T_0$ of a given tree $enc(T)$, a particular path from the root to a leaf. We call this path the *principal line* of $T_0$, denoted by $line(T_0)$, and defined as follows:

- $root(T_0)$ belongs to $line(T_0)$;
- let $v$ be an internal node of $T_0$ that belongs to $line(T_0)$, and suppose $v$ has children $v_1, v_2$. If $|tree(v_1)| \geq |tree(v_2)|$, then $v_1$ belongs to $line(T_0)$; otherwise, $v_2$ belongs to $line(T_0)$.

Validation of $enc(T)$ can be done by associating to each maximal principal line[4] an NFA that validates that particular line. We make this more precise next.

[4] A principal line is maximal if it is not included in another principal line.

*From BNTA to NFA on principal lines.* Consider the principal line $v_1 \dots v_n$ of a binary tree encoding $enc(T)$ where $v_1$ is the root and $v_n$ is a leaf. By the definition of binary encodings, each non-leaf node $v_i$ has one child $v_i'$ that does *not* belong to the principal line $v_1 \dots v_n$, for $1 \leq i < n$. Consider the sets $types(v_i')$. Note that if these sets are given, we can validate $enc(T)$ by an NFA $N$ that works on the string $v_1 \dots v_n$. For technical reasons, the constructed NFA recognizes the reverse word $v_n \dots v_1$. Essentially, the NFA guesses a sequence of state assignments to $v_n \dots v_1$ that is compatible with the transition function of $A$, given the sets of states $types(v_i')$.

The above intuition is captured as follows: We define new labels for the nodes $v_i$, that include both $\lambda(v_i)$ and the set $types(v_i')$. Thus, let $\Sigma' = \{\#\} \cup (\mathcal{P}(Q) \times \Sigma) \cup (\Sigma \times \mathcal{P}(Q))$ and $\lambda'$ be the labeling function defined as follows:

- $\lambda'(v_i) = \langle \lambda(v_i), types(v_i') \rangle$, if $v_i'$ is the right child of $v_i$, $1 \leq i < n$,
- $\lambda'(v_i) = \langle types(v_i'), \lambda(v_i) \rangle$, if $v_i'$ is the left child of $v_i$, $1 \leq i < n$.
- $\lambda'(v_n) = \lambda(v_n) = \#$.

The NFA $N$ we construct will accept the string $\lambda'(v_n) \dots \lambda'(v_1)$ iff $A = \langle \Sigma_\#, Q, Q_0, q_f, \delta \rangle$ accepts $enc(T)$. At any rate, it will compute the type derived by the sequence. More precisely, let $N = \langle \Sigma', Q, q_0, F', \delta' \rangle$, where $\Sigma'$ is as described above, $F' = \{q_f\}$, and $\delta'$ is defined by the following (and is empty everywhere else)

- $\delta'(\#, q_0) = Q_0$;
- $\delta'(\langle a, S \rangle, q) = \bigcup_{q' \in S} \delta(a, q, q')$ for $a \in \Sigma$
- $\delta'(\langle S, a \rangle, q) = \bigcup_{q' \in S} \delta(a, q', q)$ for $a \in \Sigma$

Intuitively, the NFA simulates $A$ by allowing only state transitions compatible with the transition function of $A$ and the sets of states associated to siblings. It is easy to verify that $N$ works as desired.

Note that the number of states of $N$ is $O(|Q|)$. Recall that $|Q|$ is itself $O(|\Sigma^t||d|)$ where $\tau = \langle \Sigma, \Sigma^t, d, \mu \rangle$ is the specialized DTD to which the BNTA $A$ corresponds. The size of its alphabet $\Sigma'$ is $O(|\Sigma| 2^{|Q|})$ which is $O(|\Sigma| 2^{|\Sigma^t||d|})$. Hence, each symbol in $\Sigma'$ can be represented in space $O(|\Sigma^t||d| + \log |\Sigma|)$. Notice however that our auxiliary structure never represents the alphabet or the transition mapping of $N$ explicitly.

*The auxiliary structure.* The auxiliary structure used for incremental validation includes (i) the binary tree $enc(T)$, (ii) for each subtree of $enc(T)$ its principal line and (iii) for each maximal principal line in $enc(T)$, the auxiliary transition relation tree for the NFA corresponding to that line.

Note that the principal lines can be specified concisely by annotating each node in $enc(T)$ with 0 or 1 by a labeling $\mu$ as follows: $\mu(root(enc(T))) = 0$, and for every pair of siblings $v_1, v_2$, $\mu(v_1) = 1$ and $\mu(v_2) = 0$ if $|tree(v_1)| \geq |tree(v_2)|$; otherwise, $\mu(v_1) = 0$ and $\mu(v_2) = 1$. Clearly, the principal line of a subtree $T_0$ is the unique path from $root(T_0)$ to a leaf where all non-root nodes are labeled 1. Note that the principal line of $T_0$ is maximal iff $\mu(root(T_0)) = 0$.

For example, consider the unranked tree represented in Figure 2 (top), and its binary encoding in the same figure (bottom). In the binary encoding in the figure, the nodes $w$ for which $\mu(w) = 0$ are those inside a box. Note that this identifies all maximal principal lines. The bold and underlined nodes participate in the principal line of $enc(T)$. The nodes of one of the secondary principal lines (line $j, k$) are in italics.

Part (iii) of the auxiliary structure provides the transition relation trees for the NFAs associated with the maximal principal lines. The size of each transition relation tree for an NFA $N$ is $O(|enc(T)||Q|^2)$ where $Q$ is the number of states of $N$.

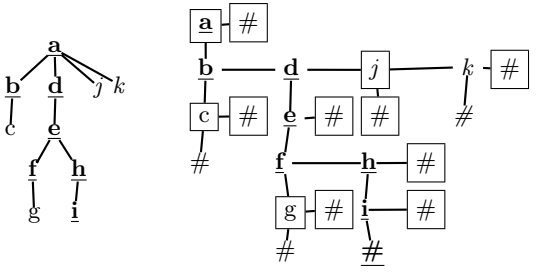



**Fig. 2.** A tree $T$ (top) and its encoding $enc(T)$

In summary, consider an input tree $T$ and a specialized DTD $\tau = \langle \Sigma, \Sigma^t, d, \mu \rangle$. In view of our construction of the BNTA $A$ from $\tau$ (Lemma 1), of the NFA $N$ from $A$ (above), and of the tree of transition relations for each NFA $N$ (Section 3) it follows that the size of the auxiliary structure associated with $T$ and $\tau$ is $O((|\Sigma^t|^2|d|^2|T|)$.

*Validation and maintenance for label renamings.* Note that label renamings in $T$ translate easily to label renamings in $enc(T)$. To validate a sequence of label renamings, it is enough to show how the auxiliary structure is maintained for each renaming in the sequence. Then validity is checked at the end using the updated auxiliary structure. So, suppose the label of some node $v$ in $enc(T)$ is modified from $a$ to $b$. Suppose first that $v$ belongs to the maximal principal line $l = v_1 \dots v_n$ of $enc(T)$, say $v = v_k$. In the string $\lambda'(v_1) \dots \lambda'(v_n)$ the label renaming corresponds to modifying the label of $v_k$ from $a$ to $b$ if $k = n$ and from $\langle a, types(v'_k) \rangle$ to $\langle b, types(v'_k) \rangle$ if $k < n$ and $v'_k$ is the right child of $v_k$ (left is analogous). Then the transition relation tree associated to $l$ is updated as in the string case in time $O(|Q|^2 \log |Q| \log |l|)$, that is $O(|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log |l|)$. Since $\log |l|$ is $O(|enc(T)|)$ and $|enc(T)|$ is $O(|T|)$, the update takes time $O(|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log |T|)$.

Now suppose that $v$ does not belong to the principal line $l$ of $enc(T)$. Then there is some $k > 0$ such that $v$ belongs to $tree(v'_k)$ where $v'_k$ is the child of some $v_k$ belonging to $l$. Note that the update to the label of $v$ may cause a change in

the value of $types(v'_k)$. In order to update $l$, we now have to first compute the new value for $types(v'_k)$, then apply the update procedure for the corresponding modification in the label $\langle \lambda(v_k), types(v'_k) \rangle$ of $v_k$. If $v$ belongs to the principal line $l'$ of $tree(v'_k)$ then the transition relation tree associated with the NFA for $l'$ can be updated as before in time $O(|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log |T|)$. This provides, in particular, the new value for $types(v'_k)$. Continuing inductively, it is clear that renaming the label of a node $v$ affects precisely the maximal principal lines encountered in the path from root to $v$. Let $M$ be the number of such maximal principal lines. Clearly, $M$ is precisely the number of nodes $w$ along the path from root to $v$ for which $\mu(w) = 0$. We next provide a bound on this number, using the notion of *line diameter* of a tree.

**Definition 3.** *Line diameter The* line diameter *of* $enc(T)$ *is the maximum number of distinct maximal principal lines crossed by any path from root to leaf in* $enc(T)$*. Equivalently, the line diameter of* $enc(T)$ *is the maximum number of nodes* $w$ *for which* $\mu(w) = 0$*, occurring along a path from root to leaf in* $enc(T)$*, where* $\mu$ *is defined as above.*

For example, the line diameter of $enc(T)$ in Figure 2 is 3. The following is easily shown.

**Lemma 2.** *The line diameter of* $enc(T)$ *is no larger than* $1 + \log |enc(T)|$.

From the bound on the line diameter of $enc(T)$, it follows that a label renaming can cause at most $O(\log |enc(T)|)$ updates to distinct transition relation trees of maximal principal lines in $enc(T)$. Since each update takes time $O(|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log |T|)$, the entire auxiliary structure can be updated in time $O(|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$. For a sequence of $m$ label renamings, updating the auxiliary structure and validating the new tree therefore takes time $O(m|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$.

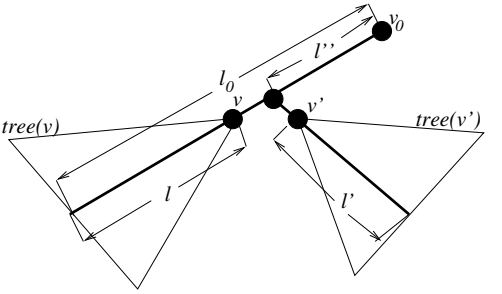



**Fig. 3.** Scenario of Line Rearrangement

*Insertions and deletions.* We next describe how to extend the maintenance and validation algorithm described above to updates that include insertions and deletions of leaf nodes.

For a maximal principal line $l$ in $enc(T)$, we denote by $N_l$ the NFA corresponding to $l$ and by $\mathcal{T}_l$ the transition relation tree corresponding to $l$ and $N_l$.

Note that each insertion or deletion of a leaf node in $T$ translates into up to four node insertions and deletions into $enc(T)$ (for example, deleting a node in $T$ may require deleting in $enc(T)$, besides the node itself, up to two leaves labeled #, and may require inserting another such leaf). This constant factor blow-up in the number of updates does not affect our analysis.

Insertions and deletions are handled by an extension of the technique used to maintain the transition relation trees for maximal principal lines in the case of label renamings. Insertions and deletions that do not cause a change in the set of maximal principal lines existing prior to the update are handled straightforwardly. More precisely, let us call an insertion or deletion *line preserving* if the restriction of $\mu$ to the nodes of $enc(T)$ that are not affected by the update is the same before and after the update. Note that an insertion may be line preserving but nonetheless introduce a new singleton maximal principal line consisting of the new node. Also observe that line-preserving updates affect precisely the maximal principal lines intersected by the path from the root of $enc(T)$ to the newly inserted node or to the parent of the deleted node. The transition relation trees for these maximal principal lines are updated as in the case of label renamings, at the same cost. If a new singleton maximal line $l$ consisting of an inserted node needs to be added, computing its auxiliary transition relation tree takes additional time $O(|Q|^2)$ where $Q$ is the set of states of the NFA $N_l$. This is dominated by the rest of the cost.

Handling inserts and deletes that are not line preserving requires more care. In this case, the set of maximal principal lines in $enc(T)$ changes as the result of updates. To illustrate the problem, consider the situation depicted in Figure 3. The maximal principal line $l_0 = \mathit{line}(\mathit{tree}(v_0))$ contains a node $v$, which has a sibling $v'$. Initially, $|\mathit{tree}(v)| \geq |\mathit{tree}(v')|$. However, a deletion in $\mathit{tree}(v)$ or an insertion in $\mathit{tree}(v')$ may make $\mathit{tree}(v')$ larger than $\mathit{tree}(v)$. In this case a new line structure is needed, where the line $l = \mathit{line}(\mathit{tree}(v))$ becomes a maximal principal line and the new principal line $\mathit{line}(\mathit{tree}(v_0))$ is the concatenation of $l''$ and $l' = \mathit{line}(\mathit{tree}(v'))$. This requires updating the auxiliary structure in two steps: First we compute the transition relation tree $\mathcal{T}_l$ for the new maximal principal line $l$ obtained by truncating $l_0$. Then we compute the transition relation tree $\mathcal{T}_{l''l'}$ for the new maximal principal line obtained by concatenating $l''$ and $l'$. Fortunately, the new transition relation trees can be computed efficiently from the old ones. Specifically, $\mathcal{T}_l$ is obtained by truncating $\mathcal{T}_{l_0}$, and $\mathcal{T}_{l''l'}$ is obtained by merging a subtree of $\mathcal{T}_{l_0}$ corresponding to $l''$ with the tree $\mathcal{T}_{l'}$. This is done by adapting usual B-tree techniques. We ommit the details of truncating and merging trees. The complexity of truncating $\mathcal{T}_{l_0}$ is $O(|Q|^2 \log |Q| \log |l_0|)$ where $Q$ is the set of states of $N_{l_0}$. The complexity of computing the merged tree $\mathcal{T}_{l''l'}$ is $O(|Q|^2 \log |Q|(\log |l'| + \log |l''|))$. Overall, the rearrangement of these lines requires which is $O(|Q|^2 \log |Q| \log |enc(T)|)$. Also, note that a single insertion or deletion may cause at most $O(\log |enc(T)|)$ line rearrangements (one for each maximal principal line intersected by the path from

root to the affected node). Thus, all line rearrangements can be done in time $O(|Q|^2 \log |Q| \log^2 |enc(T)|)$. In terms of the original specialized DTD and input tree $T$, this is $O((|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$. Once the line rearrangements have been computed, additional updates to the transition relation trees of maximal principal lines may have to be computed, as in the case of label renamings. This takes again time $O((|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$. In summary, the size of the auxiliary structure used for incremental validation is $O((|\Sigma^t|^2|d|^2|T|)$. Maintaining the auxiliary structure and validating the updated tree following a sequence of $m$ updates (label renamings, insertions, or deletions) is done in time $O(m|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$.

## 6 Conclusions and Future Work

The incremental validation algorithms we exhibited are significant improvements over brute-force validation from scratch. However, several issues need further investigation:

*Lower bounds.* To understand how close our algorithms are from optimal, it would be of interest to exhibit lower bounds on incremental maintenance of strings, DTDs, and specialized DTDs. There are known results that yield lower bounds for validation from scratch: acceptance of a tree by a tree automaton is complete for uniform $NC^1$ under DLOGTIME reductions [Loh01]. However, this does not seem to yield any non-trivial lower bound on the incremental validation problem. We are not aware of any work providing such lower bounds applicable to our framework.

*Optimizing over multiple updates.* For a sequence of $m$ updates, our incremental validation algorithm modifies the auxiliary structure one update at a time, then checks validity of the final updated tree. Clearly, it is sometimes more efficient to consider groups of updates at a time. For example, this may avoid performing unnecessary intermediate line rearrangements in the incremental algorithm for specialized DTDs. Also, if the number of updates is large compared to the size of the resulting tree, it may be more efficient to re-validate from scratch.

*More complex updates on trees.* We only considered here elementary updates affecting one node at a time. Some scenarios, such as XML editors, require more complex updates arising from manipulation of entire subtrees (deletion, insertion, cut-and-paste, etc). Our approach can still be applied by reducing each of these updates to a sequence of elementary updates. However, in this case it may be more efficient to consider updates of coarser granularity.

## References

[BKMW01] A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over non-ranked alphabets. HKUST-TCSC-2001-05, HKUST 2001. Available at http://www.cs.ust.hk/tcsc/RR/2001-05.ps.gz.

[BKW98] A. Bruggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[BM99] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *Int'l. Conf. on Database Theory*, pages 296–313, 1999.

[CDSS98] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *Proc. ACM SIGMOD*, 177–188, 1998.

[CLR] T. Cormen and C. Leiserson and R. Rivest. Introduction to Algorithms, Mc Graw-Hill, 1992.

[DS95] G. Dong and J. Su. Space-bounded foies. In *Proc. ACM PODS*, 139–150, 1995.

[GM80] C. Ghezzi and D. Mandrioli. Augmenting parsers to support incrementality. *JACM*, 27(3), 1980.

[GMUW01] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[HI02] B. Hesse and N. Immerman. Complete problems for dynamic complexity classes. *Proc.IEEE LICS*, 313–322, 2002.

[JG82] F. Jalili and J. Gallier. Building friendly parsers. In *Proc. ACM POPL*, 1982.

[Lar95] J. Larcheveque. Optimal incremental parsing. *ACM Transactions on Programming Languages and Systems*, 17(1), 1995.

[Li95] W. Li. A simple and efficient incremental LL(1) parsing. In *Theory and Practice of Informatics*, 1995.

[Lin93] G. Linden. Incremental updates in structured documents, 1993. Licentiate Thesis, Report C-1993-19, Department of Computer Science, University of Helsinki.

[Loh01] M. Lohrey. On the parallel complexity of tree automata. In *Proceedings of the 12th RTA, LNCS 2051*, 2001.

[MPS90] A. Murching, Y. Prasant, and Y. Srikant. Incremental recursive descent parsing. *Computer Languages*, 15(4), 1990.

[MSVT94] P.B. Miltersen, S. Subramanian, J.S. Vitter, and R. Tamassia. Complexity models for incremental computation. *TCS*, 130(1):203–236, 1994.

[Nev02] F. Neven. Automata, logic and XML. In *Computer Science Logic*, 2-26, 2002.

[Pet95] L. Petrone. Reusing batch parsers as incremental parsers. In *Proc. FSTTCS*, 1995.

[PI97] S. Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *JCSS*, 55(2), 1997.

[PV00] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. ACM PODS*, 35–46, 2000.

[Seg02] L. Segoufin. Personal communication, 2002.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Springer Verlag, 1999.

[W3C98] W3C. The extensible markup language (XML), 1998. W3C Recomendation available at http://www.w3c.org/XML.

[W3C01] W3C. XML schema definition, 2001. W3C Recomendation available at http://www.w3c.org/XML/Schema.

[WG98] T. Wagner and S. Graham. Efficient and flexible incremental parsing. *ACM Transactions on Programming Languages and Systems*, 20(2), 1998.

# Typechecking Top-Down Uniform Unranked Tree Transducers

Wim Martens and Frank Neven

University of Limburg
{wim.martens,frank.neven}@luc.ac.be

**Abstract.** We investigate the typechecking problem for XML queries: statically verifying that every answer to a query conforms to a given output schema, for inputs satisfying a given input schema. As typechecking quickly turns undecidable for query languages capable of testing equality of data values, we return to the limited framework where we abstract XML documents as labeled ordered trees. We focus on simple top-down recursive transformations motivated by XSLT and structural recursion on trees. We parameterize the problem by several restrictions on the transformations (deleting, non-deleting, bounded width) and consider both tree automata and DTDs as output schemas. The complexity of the typechecking problems in this scenario range from PTIME to EXPTIME.

## 1 Introduction

The emergence of XML as the likely standard for representing and exchange of data on the Web confirmed the central role of semistructered data. However, it has also marked the return of the schema. In the context of the Web, schemas can be used to validate data exchange. In a typical scenario, a user community agrees on a common schema and on producing only data conforming to that schema. This raises the issue of typechecking: verifying at compile time that every XML document which is the result of a specified query applied to a valid input document, satisfies the output schema [23,24].

Obviously, typechecking depends on the transformation language and the schema language at hand. As shown by Alon et al. [1,2], when transformation languages have the ability to compare data values, the typechecking problem quickly turns undecidable. Milo, Suciu, and Vianu argued that the capability of most XML transformation languages can be encompassed by $k$-pebble transducers when data values are ignored and XML documents can be abstracted by labeled ordered trees [15]. Further, the authors showed that the typechecking problem in this context is decidable. More precisely, given two types $\tau_1$ and $\tau_2$, represented by tree automata, and a $k$-pebble transducer $T$, it is decidable whether $T(t) \in \tau_2$ for all $t \in \tau_1$. Here, $T(t)$ is the tree obtained by running $T$ on input $t$. The complexity, however, is very bad: non-elementary.

In an attempt to lower the complexity, we consider much simpler tree transformations: those defined by deterministic top-down uniform tree transducers

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 64–78, 2003.

**Table 1.** The presented results: the top row of the table shows the representation of the input and output schemas and the left column shows the type of tree transducer.

| | NTA | DTA | DTD(NFA) | DTD(DFA) | DTD($\mathcal{SL}$) |
|---|---|---|---|---|---|
| general | EXPTIME | EXPTIME | EXPTIME | EXPTIME | EXPTIME |
| non-deleting | EXPTIME | EXPTIME | PSPACE | PSPACE | CONP |
| bounded width | EXPTIME | in EXPTIME/PSPACE-hard | PSPACE | PTIME | CONP |

on unranked trees. Such transformations correspond to structural recursion on trees [5] and to simple top-down XSLT transformations [3,6]. The transducers are called uniform as they cannot distinguish between the order of siblings. In brief, a transformation consists of a top-down traversal of the input tree where every node is replaced by a new tree (possibly the empty tree).

We show that the ability of transducers to delete interior nodes (i.e., replacing them by the empty tree) already renders the typechecking problem EXPTIME-hard for very simple DTDs (e.g. DTDs with deterministic regular expressions on their right hand side). To obtain a lower complexity, for the remainder of the paper, we focus on non-deleting transformations. Our inquiries reveal that the complexity of the typechecking problem of non-deleting transducers is determined by two features: (1) non-determinism in the schema languages; and, (2) unbounded copying of subtrees by the transducer. Only when we disallow both features, we get a PTIME-complexity for the typechecking problem. An overview of our results is given in Table 1. Unless specified otherwise, all complexities are both upper and lower bounds. The top row of the table shows the representation of the input and output schemas and the left column shows the type of tree transducer. NTA and DTA stand for non-deterministic and deterministic tree automata, respectively. Such automata abstract the expressiveness of XML Schema [7]. DTD($X$) stands for DTDs whose right-hand sides consist of regular languages in $X$. The exact definitions are given in section 2.

*Related Work.* A problem related to typechecking is type inference [14,19]. This problem consists in constructing a tight output schema, given an input schema and a transformation. Of course, solving the type inference problem implies a solution for the typechecking problem: check containment of the inferred schema into the given one. However, characterizing output languages of transformations is quite hard [19].

The transducers considered in the present paper are restricted versions of the ones studied by Maneth and Neven [13]. They already obtained a non-elementary upper bound on the complexity of typechecking (due to the use of monadic second-order logic in the definition of the transducers).

Although the structure of XML documents can be faithfully represented by unranked trees (these are trees without a bound on the number of children of nodes), Milo, Suciu, and Vianu chose to study $k$-pebble transducers over binary trees as there is an immediate encoding of unranked trees into binary ones. The top-down variants of $k$-pebble transducers are well-studied on binary trees [11].

However, these results do not aid in the quest to characterize precisely the complexity of typechecking transformations on unranked trees. Indeed, the class of unranked tree transductions can *not* be captured by ordinary transducers working on the binary encodings. Macro tree transducers can simulate our transducers on the binary encodings [13,9], but as very little is known about their complexity this observation is not of much help. For these reasons, we chose to work directly with unranked tree transducers.

Tozawa considered typechecking w.r.t. tree automata for a fragment of top-down XSLT similar to ours [25]. He adapts the backward type inference technique of [15] and obtains a double exponential time algorithm.

Due to space limitations, we only provide sketches of proofs.

## 2 Definitions

In this section we provide the necessary background on trees, automata, and uniform tree transducers.

### 2.1 Trees and Hedges

We fix a finite alphabet $\Sigma$. The set of unranked $\Sigma$-trees, denoted by $\mathcal{T}_\Sigma$, is the smallest set of strings over $\Sigma$ and the parenthesis symbols ')' and '(' such that for $\sigma \in \Sigma$ and $w \in \mathcal{T}_\Sigma^*$, $\sigma(w)$ is in $\mathcal{T}_\Sigma$. We write $\sigma$ rather than $\sigma()$. Note that there is no a priori bound on the number of children of a node in a $\Sigma$-tree; such trees are therefore *unranked*. In the following, whenever we say tree, we always mean $\Sigma$-tree. A *hedge* is a finite sequence of trees. The set of hedges, denoted by $\mathcal{H}_\Sigma$, is defined as $\mathcal{T}_\Sigma^*$.

For every hedge $h \in \mathcal{H}_\Sigma$, the *set of nodes of* $h$, denoted by $\mathrm{Dom}(h)$, is the subset of $\mathbb{N}^*$ defined as follows:

- if $h = \varepsilon$, then $\mathrm{Dom}(h) = \emptyset$;
- if $h = t_1 \cdots t_n$ where each $t_i \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(h) = \bigcup_{i=1}^{n} \{iu \mid u \in \mathrm{Dom}(t_i)\}$; and,
- if $h = \sigma(w)$, then $\mathrm{Dom}(h) = \{\varepsilon\} \cup \mathrm{Dom}(w)$.

In the sequel we adopt the following convention: we use $t, t_1, t_2, \ldots$ to denote trees and $h, h_1, h_2, \ldots$ to denote hedges. Hence, when we write $h = t_1 \cdots t_n$ we tacitly assume that all $t_i$'s are trees. For every $u \in \mathrm{Dom}(h)$, we denote by $\mathrm{lab}^h(u)$ the label of $u$ in $h$. A *tree language* is a set of trees.

### 2.2 DTDs and Tree Automata

We use extended context-free grammars and tree automata to abstract from DTDs and the various proposals for XML schemas. Further, we parameterize the definition of DTDs by a class of representations $\mathcal{M}$ of regular string languages like, e.g., the class of DFAs or NFAs. For $M \in \mathcal{M}$, we denote by $L(M)$ the set of strings accepted by $M$.

**Definition 1.** Let $\mathcal{M}$ be a class of representations of regular string languages over $\Sigma$. A DTD is a tuple $(d, s_d)$ where $d$ is a function that maps $\Sigma$-symbols to elements of $\mathcal{M}$ and $s_d \in \Sigma$ is the start symbol. For simplicity, we usually denote $(d, s_d)$ by $d$.

A tree $t$ satisfies $d$ if $\text{lab}^t(\varepsilon) = s_d$ and for every $u \in \text{Dom}(t)$ with $n$ children $\text{lab}^t(u1) \cdots \text{lab}^t(un) \in L(d(\text{lab}^t(u)))$. By $L(d)$ we denote the tree language accepted by $d$.

We parameterize DTDs by the formalism used to represent the regular language $\mathcal{M}$. Therefore, we denote by $\text{DTD}(\mathcal{M})$ the class of DTDs where the regular string languages are represented by elements of $\mathcal{M}$. The *size* of a DTD is the sum of the sizes of the elements of $\mathcal{M}$ used to represent the function $d$.

To define unordered languages we make use of the specification language $\mathcal{SL}$ inspired by [17] and also used in [1,2]. The syntax of the language is as follows.

**Definition 2.** For every $\sigma \in \Sigma$ and natural number $i$, $\sigma^{=i}$ and $\sigma^{\geq i}$ are *atomic $\mathcal{SL}$-formulas*; true is also an atomic $\mathcal{SL}$-formula. Every atomic $\mathcal{SL}$-formula is an $\mathcal{SL}$-formula and the negation, conjunction, and disjunction of $\mathcal{SL}$-formulas are also $\mathcal{SL}$-formulas.

A string $w$ over $\Sigma$ satisfies an atomic formula $\sigma^{=i}$ if it has exactly $i$ occurrences of $\sigma$; $w$ satisfies $\sigma^{\geq i}$ if it has at least $i$ occurrences of $\sigma$. Further, true is satisfied by every string.[1] Satisfaction of Boolean combinations of atomic formulas is defined in the obvious way. As an example, consider the $\mathcal{SL}$ formula $\text{co-producer}^{\geq 1} \rightarrow \text{producer}^{\geq 1}$. This expresses the constraint that a co-producer can only occur when a producer occurs. The *size* of an $\mathcal{SL}$-formula is the number of symbols that occur in it (every $i$ in $\sigma^{=i}$ or $\sigma^{\geq i}$ is written in binary notation).

We recall the definition of non-deterministic tree automata from [4]. We refer the unfamiliar reader to [16] for a gentle introduction.

**Definition 3.** A *nondeterministic tree automaton (NTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where $Q$ is a finite set of states, $F \subseteq Q$ is the set of final states, and $\delta$ is a function $Q \times \Sigma \rightarrow 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over $Q$ for every $a \in \Sigma$ and $q \in Q$.

A *run* of $B$ on a tree $t$ is a labeling $\lambda : \text{Dom}(t) \rightarrow Q$ such that for every $v \in \text{Dom}(t)$ with $n$ children, $\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v), \text{lab}^t(v))$. Note that when $v$ has no children, then the criterion reduces to $\varepsilon \in \delta(\lambda(v), \text{lab}^t(v))$. A run is *accepting* iff the root is labeled with an accepting state, that is, $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run. The set of all accepted trees is denoted by $L(B)$. We extend the definition of $\delta$ to trees and denote this by $\delta^*(t)$: if $t$ consists of only one node labeled with $a$ then $\delta^*(t) = \{q \mid \varepsilon \in \delta(q, a)\}$; if $t$ is of the form $a(t_1 \cdots t_n)$, then $\delta^*(t) = \{q \mid \exists q_1 \in \delta^*(t_1), \ldots, \exists q_n \in \delta^*(t_n)$ and $q_1 \cdots q_n \in \delta(q, a)\}$. So, $t$ is accepted if $\delta^*(t) \cap F \neq \emptyset$.

A tree automaton is *bottom-up deterministic* if for all $q, q' \in Q$ with $q \neq q'$ and $a \in \Sigma$, $\delta(q, a) \cap \delta(q', a) = \emptyset$. We denote the set of bottom-up deterministic

[1] The empty string is obtained by $\bigwedge_{\sigma \in \Sigma} \sigma^{=0}$ and the empty set by $\neg$ true.

NTAs by DTA. A tree automaton is *top-down deterministic* if for all $q, q' \in Q$ with $q \neq q'$, $a \in \Sigma$, and $n \geq 0$, $\delta(q, a)$ contains at most one string of length $n$.

Like for DTDs, we parameterize NTAs by the formalism used to represent the regular languages in the transition functions $\delta(q, a)$. So, for $\mathcal{M}$ a class of representations of regular languages, we denote by NTA($\mathcal{M}$) the class of NTAs where all transition functions are represented by elements of $\mathcal{M}$. The *size* of an automaton $B$ is then $|Q| + |\Sigma| + \sum_{q,a} |\delta(q, a)| + |F|$. Here, by $|\delta(q, a)|$ we denote the size of the automaton accepting $\delta(q, a)$. Unless explicitly specified otherwise, $\delta(q, a)$ is always represented by an NFA.

Let 2AFA be the class of two-way alternating finite automata [12]. We give without proof the following theorem which will be a useful tool for obtaining upper bounds.

**Theorem 1.** *1. Emptiness of NTA(NFA) is in* PTIME*;*
*2. Emptiness of NTA(2AFA) is in* PSPACE.

### 2.3 Transducers

We next define the tree transducers used in this paper. To simplify notation, we restrict to one alphabet. That is, we consider transductions mapping $\Sigma$-trees to $\Sigma$-trees. It is, however, possible to define transductions where the input alphabet differs from the output alphabet.

For a set $Q$, denote by $\mathcal{H}_\Sigma(Q)$ (resp. $\mathcal{T}_\Sigma(Q)$) the set of $\Sigma$-hedges (resp. trees) where leaf nodes can be labeled with elements from $Q$.

**Definition 4.** A *uniform tree transducer* is a tuple $(Q, \Sigma, q_0, R)$, where $Q$ is a finite set of states, $\Sigma$ is the input and output alphabet, $q_0 \in Q$ is the initial state, and $R$ is a finite set of rules of the form $(q, a) \to h$, where $a \in \Sigma$, $q \in Q$, and $h \in \mathcal{H}_\Sigma(Q)$. When $q = q_0$, $h$ is restricted to $\mathcal{T}_\Sigma(Q) \setminus Q$.

The restriction on rules with the initial state ensures that the output is always a tree rather than a hedge. For the remainder of this paper, when we say tree transducer, we always mean *uniform* tree transducer.

*Example 1.* Let $T = (Q, \Sigma, p, R)$ where $Q = \{p, q\}$ and $R$ contains the rules

$$\begin{array}{ll} (p, a) \to d(e) & (p, b) \to c(q\ p) \\ (q, a) \to c\ q & (q, b) \to d(q) \end{array}.$$

Our definition of tree transducers corresponds to structural recursion [5] and a fragment of top-down XSLT. For instance, the XSLT program equivalent to the above transducer is given in Figure 1 (we assume the program is started in mode $p$). □

The translation defined by $T = (Q, \Sigma, q_0, R)$ on a tree $t$ in state $q$, denoted by $T^q(t)$, is inductively defined as follows: if $t = \varepsilon$ then $T^q(t) := \varepsilon$; if $t = a(t_1 \cdots t_n)$ and there is a rule $(q, a) \to h \in R$ then $T^q(t)$ is obtained from $h$ by replacing

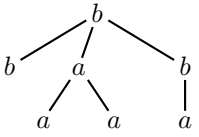
```
<xsl:template match="a" mode ="p">
  <d>
     <e/>
  </d>
</xsl:template>

<xsl:template match="a" mode ="q">
  <c/>
  <xsl:apply-templates mode="q"/>
</xsl:template>

<xsl:template match="b" mode ="p">
  <c>
     <xsl:apply-templates mode="q"/>
     <xsl:apply-templates mode="p"/>
  </c>
</xsl:template>

<xsl:template match="b" mode ="q">
  <d>
     <xsl:apply-templates mode="q"/>
  </d>
</xsl:template>
```

**Fig. 1.** The XSLT program equivalent to the transducer of Example 1.

every node $u$ in $h$ labeled with $p$ by the hedge $T^p(t_1) \cdots T^p(t_n)$. Note that such nodes $u$ can only occur at leaves. So, $h$ is only extended downwards. If there is no rule $(q, a) \to h \in R$ then $T^q(t) := \varepsilon$. Finally, define the transformation of $t$ by $T$, denoted by $T(t)$, as $T^{q_0}(t)$.

For $a \in \Sigma$, $q \in Q$ and $(q, a) \to h \in R$, we denote $h$ by $\mathrm{rhs}(q, a)$. If $q$ and $a$ are not important, we say that $h$ is a rhs. The *size* of $T$ is $|Q|+|\Sigma|+\sum_{(q,a)} |\mathrm{rhs}(q, a)|$.

*Example 2.* In Figure 2 we give the translation of the tree $t$ defined as




by the transducer of Example 1. □

We discuss two features which are of importance in the remainder of the paper: copying and deleting. The rule $(p, b) \to c(q\,p)$ in the above example copies the children of the current node in the input tree two times: one copy is processed in state $q$ and the other in state $p$. The symbol $c$ is the parent node of the two copies. So the current node in the input tree corresponds to the latter node. The rule $(q, a) \to c\,q$ also copies the children of the current node two times. However, in this case, one copy is replaced by the single symbol tree $c$, the other copy is obtained by processing the children in state $q$. No parent node is given for this copy. So, there is no corresponding node for the current node in the input tree. We, therefore, say it is deleted. For instance, $T^q(a(b)) = c\,d$ where $d$ corresponds to $b$ and not to $a$.

### 2.4 The Typechecking Problem

We define the problem central to this paper.

**Definition 5.** A tree transducer $T$ *typechecks* w.r.t. to an input tree language $S_{\text{in}}$ and an output tree language $S_{\text{out}}$, if $T(t) \in S_{\text{out}}$ for every $t \in S_{\text{in}}$.

We parameterize the typechecking problem by the kind of tree transducers and tree languages we allow. Let $\mathcal{T}$ be a class of transducers and $\mathcal{S}$ be a class of tree languages. Then TC$[\mathcal{T}, \mathcal{S}]$ denotes the typechecking problem where $T \in \mathcal{T}$ and $S_{\text{in}}, S_{\text{out}} \in \mathcal{S}$. The size of the input of the typechecking problem is the sum of the sizes of the input and output schema and the tree transducer.

A transducer $T$ has *width* $k$ if there are at most $k$ occurrences of states in every rhs of $T$. By $\mathcal{BW}_k$ we denote the class of transducers of width $k$. A transducer is *non-deleting* if no states occur at the top-level of a rhs. We denote by $\mathcal{T}_g$ the class of all transducers and by $\mathcal{T}_{nd}$ the class of non-deleting transducers. For a class of representations of regular string languages $\mathcal{M}$, we write TC$[\mathcal{T},\mathcal{M}]$ rather than TC$[\mathcal{T}$,DTD$(\mathcal{M})]$.

## 3 The General Case

When we do not restrict our transducers in any way, the typechecking problem is in EXPTIME and is EXPTIME-hard for even the simplest DTDs: those where the right-hand sides are specified with $\mathcal{SL}$-formulas or with DFAs. The main reason is that the deleting states allow the transducer to simulate deterministic top-down tree automata in such a way that the transducer produces no output besides acceptance information. In this way, even a very simple DTD can check whether the output was rejected or not. By copying the input several times, we can execute several deterministic tree automata in parallel. These are all the ingredients we need for a reduction from non-emptiness of the intersection of an arbitrary number of deterministic tree automata which is known to be EXPTIME-hard.

**Theorem 2.** *1. TC[$\mathcal{T}_g$,NTA] is in* EXPTIME*;*
*2. TC[$\mathcal{T}_g$,$\mathcal{SL}$] is* EXPTIME*-hard;*
*3. TC[$\mathcal{T}_g$,DFA] is* EXPTIME*-hard.*

*Proof.* (Sketch) (1) Let $T = (Q_T, \Sigma, q_T^0, R_T)$ be a transducer and let $A_{\text{in}}$ and $A_{\text{out}} = (Q_A, \Sigma, \delta_A, F_A)$ be two NTAs representing the input and output schema, respectively. We next describe a *non*-deleting transducer $S$ and an NTA $B_{\text{out}}$ which can be constructed in LOGSPACE, such that $T$ typechecks w.r.t. $A_{\text{in}}$ and $A_{\text{out}}$ iff $S$ typechecks w.r.t. $A_{\text{in}}$ and $B_{\text{out}}$. From Theorem 3(1) it then follows that TC[$\mathcal{T}_g$,NTA] is in EXPTIME.

Intuitively, $S$ puts a # whenever $T$ would process a deleting state. For instance, the rule $(q, a) \to c\, q$ is replaced by $(q, a) \to c\#(q)$. We introduce some notation to characterize the behavior of $B_{\text{out}}$. Define the #-eliminating function $\gamma$ as follows: $\gamma(\sigma(h))$ is $\gamma(h)$ when $\sigma = \#$ and $\sigma(\gamma(h))$ otherwise; further, $\gamma(t_1 \cdots t_n) := \gamma(t_1) \cdots \gamma(t_n)$. Then, clearly, for all $t \in \mathcal{T}_\Sigma$, $T(t) = \gamma(S(t))$. $B_{\text{out}}$ then accepts a tree $t$ iff $\gamma(t) \in L(A_{\text{out}})$.

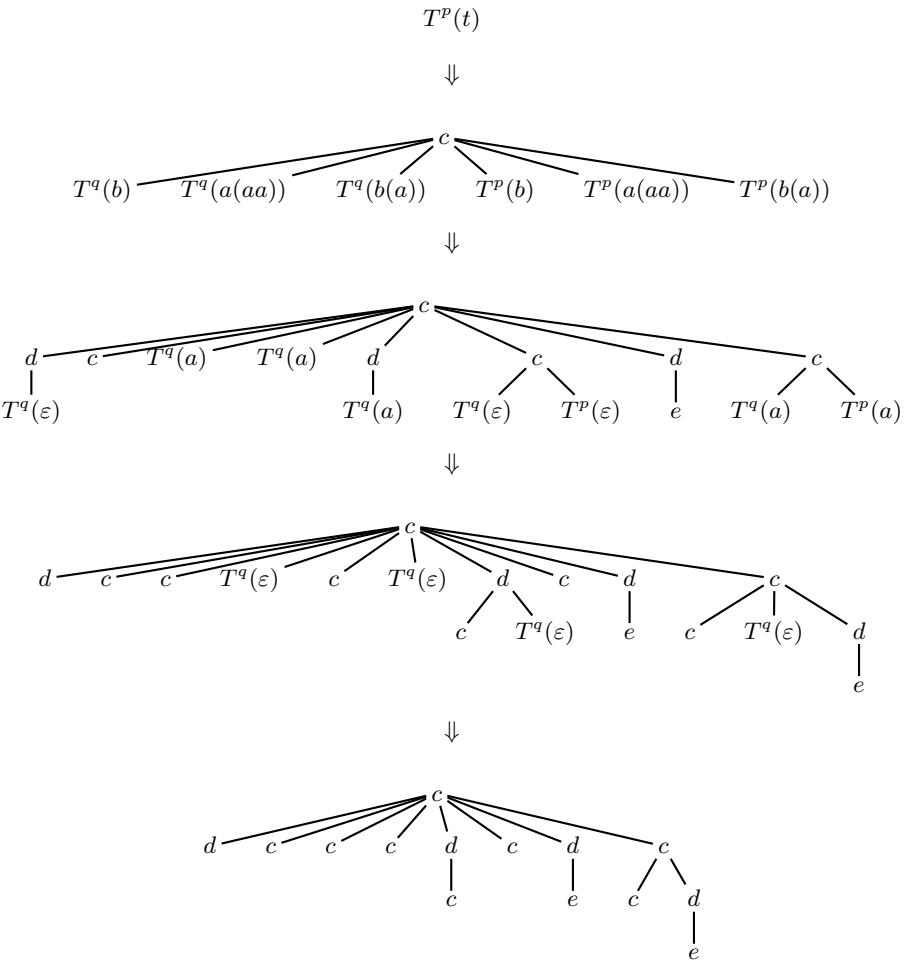



**Fig. 2.** The translation of $t = b(b\,a(a\,a)b(a))$ by the transducer $T$ of Example 1.

(2) We use a reduction from the intersection problem of deterministic binary top-down tree automata $A_i$ $(i = 1 \dots n)$, which is known to be hard for EXPTIME [21]. The problem is stated as follows, given deterministic binary top-down automata $A_1, \dots, A_n$, is $\bigcap_{i=1}^{n} L(A_i) = \emptyset$? We define a transducer $T$ and two DTDs $d_{\text{in}}$ and $d_{\text{out}}$ such that $\bigcap_{i=1}^{n} L(A_i) \neq \emptyset$ iff $T$ does not typecheck w.r.t. $d_{\text{in}}$ and $d_{\text{out}}$. In the construction, we exploit the copying power of transducers to make $n$ copies of the input tree: one for each $A_i$. By using deleting states, we can execute each $A_i$ on its copy of the input tree without producing output. When an $A_i$ does not accept, we output an *error* symbol under the root of the output tree. The output DTD should then only check that an *error* symbol always appears.

The proof of (3) is similar to the one for (2). □

## 4 Non-deleting Transformations

In an attempt to lower the complexity, we consider, in the present section, non-deleting transformations w.r.t. various schema formalisms. We observe that when schemas are represented by tree automata, the complexity remains EXPTIME-hard. When tree languages are represented by DTDs, the complexity of the typechecking problem drops to PSPACE and is hard for PSPACE even when right-hand sides of rules are represented by DFAs. The main reason for this is that the tree transducers can still make an unbounded number of copies of the input tree. This allows to simulate in parallel an unbounded number of DFAs and makes it possible to reduce the intersection emptiness problem of DFAs to the typechecking problem. In the next section, we therefore constrain this copying power. In summary, we prove the following results:

**Theorem 3.** *1. TC[$\mathcal{T}_{nd}$,NTA] is* EXPTIME*-complete;*
*2. TC[$\mathcal{T}_{nd}$,DTA] is* EXPTIME*-complete;*
*3. TC[$\mathcal{T}_{nd}$,NFA] is* PSPACE*-complete;*
*4. TC[$\mathcal{T}_{nd}$,DFA] is* PSPACE*-complete;*
*5. TC[$\mathcal{T}_{nd}$,$\mathcal{SL}$] is* CONP*-complete.*

### 4.1 Tree Automata

Consider the case where the input and output schemas are represented by NTA(NFA)s. One way to obtain an appropriate typechecking algorithm, would be to build a composite automaton $A_{\text{co}}$ that on input $t$, runs $A_{\text{out}}$ on $T(t)$ without actually constructing $T(t)$. The given instance typechecks iff $L(A_{\text{in}}) \subseteq L(A_{\text{co}})$. However, constructing $A_{\text{co}}$ would lead to an exponential blowup because the state set of $A_{\text{co}}$ would be $2^{Q_T} \times 2^{Q_{\text{out}}}$. Since $A_{\text{co}}$ is nondeterministic because $A_{\text{out}}$ is nondeterministic, solving the inclusion problem on this instance would lead to a double exponential time algorithm. Thus, to show that TC[$\mathcal{T}_{nd}$,NTA] can be solved in EXPTIME, we need a slightly more sophisticated approach.

**Theorem 3(1).** *TC[$\mathcal{T}_{nd}$,NTA] is* EXPTIME*-complete.*

*Proof.* (Sketch) Hardness is immediate as containment of NTAs is already hard for EXPTIME [20]. We, therefore, only prove membership in EXPTIME. The proof is similar in spirit to a proof in [18], which shows that containment of Query Automata is in EXPTIME. Let $T = (Q_T, \Sigma, q_T^0, R_T)$ be a non-deleting tree transducer and let $A_{\text{in}} = (Q_{in}, \Sigma, \delta_{in}, F_{in})$ and $A_{\text{out}} = (Q_{out}, \Sigma, \delta_{out}, F_{out})$ be the NTAs representing the input and output schema, respectively.

For ease of exposition, we restrict hedges in the rhs of $T$ to be trees. In brief, our algorithm computes the set

$$P = \{(S, f) \mid S \subseteq Q_{in}, f : Q_T \to 2^{Q_{out}}, \exists t \text{ such that } S = \delta^*_{in}(t) \text{ and } \forall q \in Q_T, f(q) = \delta^*_{out}(T^q(t))\}.$$

$P_0 := \emptyset$;
$i := 1$;
$P_1 := \{(S, f) \mid \exists a : \forall r \in S : \varepsilon \in \delta_{in}(r, a), \forall q \in Q_T : f(q) = \delta^*_{out}(T^q(a))\}$;
**while** $P_i \neq P_{i-1}$ **do**
  $P_i := \big\{(S, f) \mid \exists (S_1, f_1) \cdots (S_n, f_n) \in P^*_{i-1}, \exists a \in \Sigma :$
    $S = \{p \mid \exists r_k \in S_k, k = 1 \ldots n, r_1 \cdots r_n \in \delta_{in}(p, a)\}$
    $\forall q \in Q_T : f(q) = \delta^*_{out}\big(\mathrm{rhs}(q, a)[p \leftarrow f_1(p) \cdots f_n(p) \mid p \in Q_T]\big)\big\}$;
  $i := i + 1$;
**end while**
$P := P_i$;

**Fig. 3.** The algorithm of Theorem 3(1) computing $P$.

Intuitively, in the definition of $P$, $t$ can be seen as a witness of $(S, f)$. That is, $S$ is the set of states reachable by $A_{\text{in}}$ at the root of $t$, while for each state $q$ of the transducer, $f(q)$ is the set of states reachable by $A_{\text{out}}$ at the root of $T^q(t)$ (recall that this is the translation of $t$ started in state $q$). So, the given instance does *not* typecheck iff there exists an $(S, f) \in P$ such that $F_{in} \cap S \neq \emptyset$ and $F_{out} \cap f(q^0_T) = \emptyset$. In Figure 3, an algorithm for computing $P$ is depicted. By $\mathrm{rhs}(q, a)[p \leftarrow f_1(p) \cdots f_n(p) \mid p \in Q_T]$, we denote the tree obtained from $\mathrm{rhs}(q, a)$ by replacing every occurrence of a state $p$ by the sequence $f_1(p) \cdots f_n(p)$. For $c \in \{in, out\}$, $\delta^*_c : \mathcal{T}_\Sigma(2^{Q_c}) \to 2^{Q_c}$ is the transition function extended to trees in $\mathcal{T}_\Sigma(2^{Q_c})$. To be precise, for $a \in \Sigma$, $\delta^*_c(a) := \{q \mid \varepsilon \in \delta_c(q, a)\}$; for $P \subseteq Q_c$, $\delta^*_c(P) := P$; and, $\delta^*_c(a(t_1 \cdots t_n)) := \{q \mid \exists q_i \in \delta^*_c(t_i) : q_1 \cdots q_n \in \delta^*_c(q, a)\}$. The correctness of the algorithm follows from the following lemma which can be easily proved by induction.

**Lemma 1.** *A pair $(S, f)$ has a witness tree of depth $i$ iff $(S, f) \in P_i$.*

It remains to show that the algorithm is in EXPTIME. The set $P_1$ can be computed in time polynomial in the sizes of $A_{\text{in}}$, $A_{\text{out}}$, and $T$. As $P_i \subseteq P_{i+1}$ for all $i$, the loop can only make an exponential number of iterations. It, hence, suffices to show that each iteration can be done in EXPTIME. Actually, we argue that it can be checked in PSPACE whether a tuple $(S, f) \in P_i$. Indeed, the question whether there are $(S_1, f_1) \cdots (S_n, f_n) \in P^*_{i-1}$ can be reduced to the emptiness test of a 2AFA $A$ which works on strings over the alphabet $P_{i-1}$. On input $(S_1, f_1) \cdots (S_n, f_n)$ the 2AFA $A$ operates as follows: for every $p \in S$ it checks whether there are $r_i \in S_i$ such that $r_1 \ldots r_n \in \delta(p, a)$. This can be done by $|S|$ traversals through the input string. Next, $A$ checks for every $q \in Q_{in} \setminus S$ whether for all $r_i \in S_i$, $r_1 \ldots r_n \notin \delta(q, a)$. This can be done by $|Q_{in} \setminus S|$ traversals through the input string while using alternation. In a similar way $f(q)$ is checked. The automaton $A$ is exponential in the input. However, we can construct $A$ on the fly when executing the PSPACE algorithm for non-emptiness. The latter algorithm is an adaptation of the technique used by Vardi [26]. As there are only exponentially many tuples $(S, f)$, the overall algorithm is in EXPTIME. □

In the remainder of this section, we examine what happens when tree automata are restricted to be deterministic. From the above result, it is immediate that TC[$\mathcal{T}_{nd}$,DTA] is in EXPTIME. To show that it is hard, we can use a reduction from the intersection problem of deterministic binary top-down tree automata like in the proof of Theorem 2(2). The reduction is almost identical to the one in Theorem 2(2): $A_{\text{in}}$ defines the same set of trees as $d_{\text{in}}$ does with the exception that $A_{\text{in}}$ enforces an ordering of the children. The transducer in the proof of Theorem 2(2) starts the in parallel simulation of the $n$ automata, but then, using deleting states, delays the output until it has reached the leaves of the input tree. In the present setting, we can not use deleting states. Instead, we copy the input tree and attach error-symbols to the leaves when an automaton rejects. The output automaton then checks whether at least one error occurred. We obtain the following theorem.

**Theorem 3(2)** *TC[$\mathcal{T}_{nd}$,DTA] is* EXPTIME*-complete.*

### 4.2 DTDs

When we consider DTDs as input schemas the complexity drops to PSPACE and CONP.

**Theorem 3(3)** *TC[$\mathcal{T}_{nd}$,NFA] is* PSPACE*-complete.*

*Proof.* (Sketch) The hardness result is immediate as containment of regular expressions is known to be PSPACE-hard [22]. For the other direction, let $T$ be a non-deleting tree transducer. Let $d_{\text{in}}$ and $d_{\text{out}}$ be the input and output DTDs, respectively. We construct an NTA(2AFA) $B$ such that $L(B) = \{t \in L(d_{\text{in}}) \mid T(t) \notin d_{\text{out}}\}$. Moreover, the size of $B$ is polynomial in the size of $T$, $d_{\text{in}}$, and $d_{\text{out}}$. Thus, $L(B) = \emptyset$ iff $T$ typechecks w.r.t. $d_{\text{in}}$ and $d_{\text{out}}$. By Theorem 1(2), the latter is in PSPACE. To explain the operation of the automaton we introduce the following notion: let $q$ be a state of $T$ and $a \in \Sigma$ then define $q(a) := z$ where $z$ is the concatenation of the labels of the top most nodes of $\text{rhs}(q, a)$. For a string $w := a_1 \cdots a_n$, we define $p(w) := p(a_1) \cdots p(a_n)$. Intuitively, the automaton $B$ now works bottom-up as follows: (1) $B$ checks that $t \in L(d_{\text{in}})$; (2) at the same time, $B$ guesses a node $v$ labeled $\sigma$ with $n$ children and picks a state $q$ in which $v$ is processed: $B$ then accepts if $h$ does not satisfy $d_{\text{out}}$, where $h$ is obtained from $\text{rhs}(\sigma, q)$ by replacing every state $p$ by $p(\text{lab}^t(u1) \cdots \text{lab}^t(un))$. As $d_{\text{out}}$ is specified by NFAs and we have to check that $d_{\text{out}}$ is *not* satisfied, we need alternation to specify the transition function of $B$. Additionally, as $T$ can copy its input, we need two-way automata.

Formally, let $T = (Q_T, \Sigma, q_0^T, \delta_T)$. Define $B = (Q^B, \Sigma, F^B, \delta^B)$ as follows. The set of states $Q^B$ is the union of the following sets: $\Sigma$, $\{(\sigma, q) \mid q \in Q_T, \sigma \in \Sigma\}$, and $\{(\sigma, q, \text{check}) \mid q \in Q_T, \sigma \in \Sigma\}$. If there is an accepting run on a tree $t$, then a node $v$ labeled with a state of the form $\sigma$, $(\sigma, q)$, $(\sigma, q, \text{check})$ has the following meaning:

$\sigma$**:** the current node is labeled with $\sigma$ and the subtree rooted at this node satisfies $d_{\text{in}}$.

$(\sigma, q)$**:** same as in previous case with the following two additions: (1) $v$ is processed by $T$ in state $q$; and, (2) a descendant of $v$ will produce a tree that will not satisfy $d_{\text{out}}$.

$(\sigma, q, \text{check})$**:** same as the previous case only now $v$ itself will produce a tree that does not satisfy $d_{\text{out}}$.

The set of final states is defined as follows: $F^B := \{(\sigma, q_0^T) \mid \sigma \in \Sigma\}$. The transition function is defined as follows:

1. $\delta^B(a, b) = \delta^B((a, q), b) = \delta^B((a, q, \text{check}), b) = \emptyset$ for all $a \neq b$;
2. $\delta^B(a, a) = d_{\text{in}}(a)$ and

$$\delta^B((a, q), a) = \{\Sigma^*(b, p)\Sigma^* + \Sigma^*(b, p, \text{check})\Sigma^* \mid p \text{ occurs in } \text{rhs}(q, a), b \in \Sigma\},$$

for all $a \in \Sigma$ and $q \in Q_T$.

3. Finally, $\delta^B((a, q, \text{check}), a) = \{a_1 \cdots a_n \mid h \notin L(d_{\text{out}}) \text{ and } a_1 \cdots a_n \in d_{\text{in}}(a)\}$. Here, $h$ is obtained from $\text{rhs}(q, a)$ by replacing every $q$ by $q(a_1 \cdots a_n)$.

We are left with the proof that $\delta^B((a, q, \text{check}), a)$ can be computed by a 2AFA $A$ with only a polynomial blowup. Before we define $A$, we define some other automata. First, for every $b \in \Sigma$, let $A_b$ be the NFA accepting $d_{out}(b)$.

For every $v$ in $\text{rhs}(q, a)$, let $w$ be the largest string in $(\Sigma \cup Q_T)^*$ such that $\text{lab}^h(v)(w)$ is a subtree rooted at $v$ in $h$. Define the 2NFA $B_v$ as follows: suppose $w$ is of the form $z_0 p_1 z_1 \cdots p_\ell z_\ell$, then $a_1 \cdots a_n \in L(B_v)$ if and only if $z_0 p_1 (a_1 \cdots a_n) z_1 \cdots p_\ell (a_1 \cdots a_n) z_\ell \in L(A_{\text{lab}^h(v)})$. As $w$ is fixed, $B_v$ can recognize this language by reading $a_1 \cdots a_n$ $\ell$ times while simulating $A_{\text{lab}^h(v)}$. Intuitively, the automaton simulates $A_{\text{lab}^h(v)}$ on $z_{i-1} p_i (a_1 \cdots a_n)$ on the $i$th pass.

It remains to describe the construction of $A$. To this end, let $A^a_{in}$ be the NFA such that $d_{\text{in}}(a) = L(A^a_{in})$. On input $a_1 \cdots a_n$, $A$ first checks whether $a_1 \cdots a_n \in L(A^a_{in})$ by simulating $A^a_{in}$. After this, $A$ goes back to the beginning of the input string, guesses an internal node $u$ in $\text{rhs}(q, a)$ and simulates the negation of $B_u$. As $B_u$ is a 2NFA, $A$ is a 2AFA. □

The intersection problem of deterministic finite automata is known to be PSPACE-hard [10] and is easily reduced to TC[$\mathcal{T}_{nd}$,DFA]. This implies the following result:

**Theorem 3(4)** *TC[$\mathcal{T}_{nd}$,DFA] is* PSPACE*-complete.*

Using $\mathcal{SL}$-expressions to define right-hand sides of DTDs reduces the complexity of typechecking to CONP.

**Theorem 3(5)** *TC[$\mathcal{T}_{nd}$,$\mathcal{SL}$] is* CONP*-complete.*

*Proof.* (Sketch) First, we prove the hardness result. Let $\phi$ be a SAT-formula and let $v_1, \ldots, v_n$ be the variables occurring in $\phi$. We define the typechecking instance

as follows. $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. We only define $d_{\text{in}}$ and $d_{\text{out}}$ for $\sigma_1$, since this is all we require. $d_{\text{in}}(\sigma_1) = \phi'$, where $\phi'$ is the formula $\phi$ with every occurrence of $v_i$ replaced by $\sigma_i^{=1}$ for $i = 1 \ldots n$. The transducer $T$ is the identity, and $d_{out}(\sigma_1) = \emptyset$. Hence, this instance typechecks iff $\phi$ is *not* satisfiable.

To prove the upper bound, let $T = (Q_T, \Sigma, q_T^0, R_T)$ and let $(d_{\text{in}}, s_{\text{in}})$ and $(d_{\text{out}}, s_{\text{out}})$ be the input and output DTD respectively. We describe an NP algorithm that accepts iff the given instance does *not* typecheck.

We introduce some notation. For a DTD $(d, s_d)$ and $\sigma \in \Sigma$, we denote by $d^\sigma$ the DTD $d$ with start symbol $\sigma$, that is, $(d, \sigma)$. Let $k$ be the largest number occurring in an $\mathcal{SL}$-formula in $d_{\text{in}}$. Set $r = (k+1) \times |\Sigma|$.

The algorithm consists of three main parts:

1. First, we sequentially guess a subset $L$ of the derivable symbols $\{b \in \Sigma \mid L(d_{\text{in}}^b) \neq \emptyset\}$.
2. Next, we guess a path of a tree in $d_{\text{in}}$. In particular, we guess a sequence of pairs $(a_i, q_i) \in L \times Q_T$, $i = 0, \ldots, m$, with $m \leq |\Sigma| \times |Q_T|$, such that
   a) $a_0 = s_{\text{in}}$ and $q_0 = q_T^0$;
   b) $\exists t, \exists u \in \text{Dom}(t)$ such that $a_0 \cdots a_m$ is the concatenation of the labels of the nodes on the path from the root to $u$; and,
   c) $\forall i = 1, \ldots, m$: $T$ visits $a_i$ in state $q_i$.
3. Finally, we guess a string $w \in L^*$ of length at most $r$ such that $T^{q_m}(a_m(w)) \notin L(d_{\text{out}}^\sigma)$ with $\sigma$ the root symbol of $T^{q_m}(a_m(w))$.

All guesses can be done at once and can be checked by a polynomial verifier. This completes the description of the algorithm. □

## 5 Transducers of Bounded Width

When we put a bound on the width (or copying power, recall the discussion at the end of sections 2.3 and 2.4) of transducers we get a PTIME algorithm for typechecking when the right-hand sides of DTDs are represented by DFAs. All other results have the same complexity as in the case of unrestricted copying.

**Theorem 4.** *1. TC[$\mathcal{BW}_k$,NTA] is* EXPTIME*-complete;*
*2. TC[$\mathcal{BW}_k$,RE] is* PSPACE*-complete;*
*3. TC[$\mathcal{BW}_k$,DFA] is* PTIME*-complete;*
*4. TC[$\mathcal{BW}_k$,$\mathcal{SL}$] is* CONP*-complete.*

The lower bounds of (1), (2), and (4) follow immediately from the construction in the proofs of Theorem 3(1), (3), and (5).

**Theorem 4(3)** *TC[$\mathcal{BW}_k$,DFA] is* PTIME*-complete.*

*Proof.* (Sketch) In the proof of Theorem 3(3), TC[$\mathcal{T}_{nd}$,NFA] is reduced to the emptiness of NTA(2AFA)s. Alternation was needed to express negation of NFAs; two-wayness was needed because $T$ could make arbitrary copies of the input tree. However, when transducers can make only a bounded number of copies and DFAs are used, TC[$\mathcal{BW}_k$,DFA] can be LOGSPACE-reduced to emptiness of NTA(NFA)s. From Theorem 1(1), it then follows that TC[$\mathcal{BW}_k$,DFA] is in PTIME. A PTIME lower bound is obtained by a reduction from PATH SYSTEMS [8]. □

## 6 Conclusion

Motivated by structural recursion and XSLT, we studied typechecking for top-down XML transformers in the presence of both DTDs and tree automata. In this setting the complexity of the typechecking problem ranges from PTIME to EXPTIME. In particular, the PTIME algorithm is obtained by restricting to non-deleting tree transducers of bounded width and DTD(DFA)s. The main open question for future research is how these restrictions can be relaxed while still having a PTIME algorithm. Another question we left open is the exact complexity of TC[$\mathcal{BW}_k$,DTA] which is in EXPTIME and PSPACE-hard.

## References


1. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. Typechecking XML views of relational databases. In *Proc. 16th IEEE Symposium on Logic in Computer Science (LICS 2001)*, pages 421–130, 2001.
2. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 560–572, 2001.
3. G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.
4. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
5. P. Buneman, M. Fernandez, and D. Suciu. UnQl: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1):76–110, 2000.
6. James Clark. XSL transformations version 1.0. http://www.w3.org/TR/WD-xslt, august 1999.
7. World Wide Web Consortium. XML Schema. http://www.w3.org/XML/Schema.
8. S.A. Cook. An observation on time-storage trade-off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974.
9. J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 1985.
10. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
11. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.
12. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, 1984.
13. S. Maneth and F. Neven. Structured document transformations based on XSL. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2000.
14. T. Milo and D. Suciu. Type inference for queries on semistructured data. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 215–226. ACM Press, 1999.

15. T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22. ACM Press, 2000.
16. F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3), 2002.
17. F. Neven and T. Schwentick. XML schemas without order. Unpublished manuscript, 1999.
18. F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.
19. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 35–46. ACM Press, 2001.
20. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
21. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
22. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 1–9, Austin, Texas, 30 April–2 May 1973.
23. D. Suciu. Typechecking for semistructured data. In *Proceedings of the 8th Workshop on Data Bases and Programming Languages (DBPL 2001)*, 2001.
24. D. Suciu. The XML typechecking problem. *SIGMOD Record*, 31(1):89–96, 2002.
25. A. Tozawa. Towards static type checking for XSLT. In Proceedings of ACM Symposium on Document Engineering, 2001.
26. M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30:261–264, March 1989.

# Structural Properties of XPath Fragments

Michael Benedikt[1], Wenfei Fan[1⋆], and Gabriel M. Kuper[2]

[1] Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07970, USA,
{benedikt,wenfei}@research.bell-labs.com
[2] Università di Trento, Via Sommarive 14, 38050 Povo, Trento, Italy, kuper@acm.org

**Abstract.** We study structural properties of each of the main sublanguages of XPath [8] commonly used in practice. First, we characterize the expressive power of these language fragments in terms of both logics and tree patterns. Second, we investigate closure properties, focusing on the ability to perform basic Boolean operations while remaining within the fragment. We give a complete picture of the closure properties of these fragments, treating XPath expressions both as functions of arbitrary nodes in a document tree, and as functions that are applied only at the root of the tree. Finally, we provide sound and complete axiom systems and normal forms for several of these fragments. These results are useful for simplification of XPath expressions and optimization of XML queries.

## 1 Introduction

XPath [8] is a language for specifying the selection of element nodes within XML documents. It has been widely used in XML query languages (e.g., XSLT [7], XQL [21], XQuery [5]), XML specifications (e.g., XML Schema [22]), and subscription systems (e.g., [6]). It supports a number of powerful modalities and thus is rather expensive to process [3,17,10]. In practice, many applications do not need the excessive power of the full language; they use only a fragment of XPath. For example, XML Schema specifies integrity constraints with an XPath fragment that does not support upward modalities (the parent and ancestor axes). It is thus necessary to study the expressiveness and optimization of these XPath fragments, since their analysis and simplification are critical for efficient processing of XML documents.

These considerations motivate us to study a variety of fragments, focusing on several dichotomies:

- **downward vs. upward:** some fragments support both downward and upward navigation, while others allow only downward traversal.
- **recursive vs. nonrecursive:** some fragments allow navigation along the ancestor and descendant axes, while others permit only parent and child axes.

⋆ Supported in part by NSF Career Award IIS-0093168; on leave from Temple Univ.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 79–95, 2003.

– **qualified vs. non-qualified:** fragments may or may not include *qualifiers* (predicates testing properties of another expression).

Our aim is to show how these fragments differ from one another, in terms of expressiveness and structural properties, and develop methods for simplifying expressions in each fragment.

Our first contribution is a characterization of expressiveness of these fragments in terms of logics as well as *tree patterns*, a concept important in many areas of computer science [12,13] and studied recently in connection with LDAP and XML [1]. Extending the preliminary results of [14], we show that the natural XPath fragments with qualifiers can be characterized via the positive existential fragment of first-order logic, and that these fragments match exactly the queries formed using appropriate tree patterns. One surprising consequence of our logical characterization is that *equality qualifiers* can be added to the fragments with qualifiers, without increase in expressive power.

The second contribution is to outline containments that hold between the fragments, and to discover what additional operations can be defined within them. Using our logical and pattern characterizations, we show that the containments holding between XPath fragments can differ significantly depending on whether their expressions are to be evaluated at arbitrary nodes of an XML document tree or are restricted to the root of the tree; that is, we describe the containments that hold under *general equivalence* and *root equivalence*. In the process we show that every expression with upward modalities is root-equivalent to one without upward navigation. This is important for, among other things, processing streaming data.

Our third contribution is the study of *closure properties* of XPath fragments. Knowing that a fragment $F$ is closed under a set of operations $S$ is helpful for optimization – manipulations involving $S$ can be done while remaining in the fragment – and is useful for XPath implementers, as it indicates that the operations in $S$ can be built on top of the fundamental operations of the fragment. We give a complete picture of the closure properties under Boolean operations for all of our fragments, making use of the logical and pattern characterizations. As with containment, we show that the closure properties of a fragment under general equivalence may differ from those under root equivalence.

The final contribution of the paper is in connection with simplification of XPath expressions. We approach this problem based on *proof systems* for XPath using a set of *axioms* that allow simplification of expressions. This is an initial step toward establishing an algebraic framework both for directing the optimization process, and for allowing an optimizer to trade off weaker simplification for lower optimization time. Note that each individual fragment requires a separate analysis of the axiomatizability of containment/equivalence. Indeed, given fragments $F_1 \subset F_2$, $F_2$ may admit a simple set of simplification rules although $F_1$ does not, due to the lack of certain closure properties in $F_1$, and vice versa.

To this end, we show preliminary results, both positive and negative, for axiomatizability of XPath fragments. We show that no finite axiomatization can exist for any of our fragments, but give sound and complete computable axiom

systems for the downward non-qualified fragments. We show that some fragments actually become finitely axiomatizable when the labels are restricted to a fixed finite alphabet. We also provide *normal forms* for some of these fragments, unique up to equivalence of expressions.

Taken together, these results give a picture of the advantages and disadvantages of working within a particular fragment.

**Related work:** There has been considerable work on expressiveness and complexity of XPath. One line of investigation studies formal models of XPath [3, 17,16,15], abstracting away from the concrete syntax of XPath into a powerful automaton model – complexity bounds are then derived from bounds on the automata. The results to date have given considerable insight into the expressiveness and complexity of XPath as a whole, but have shed little light on the distinction between one fragment of XPath and another. To the best of our knowledge, there is no previous work dealing directly with the expressiveness of XPath fragments.

An active area of research with direct bearing on XPath optimization is the analysis of the *containment problem* for XPath: in a series of papers [1,9,14,24], lower and upper bounds for the complexity have been established for a number of XPath fragments. Note that understanding axiomatizability of containment in a fragment requires a fundamentally different analysis from the semantic methods used in these papers, since the existence of an axiom system is a property of the fragment *as a whole*. Solving containment itself is only a step toward optimization, but developing a framework for, and thorough study of, simplification in the context of XPath fragments remain an important open research issue.

Closest in spirit to our paper is [18]. It presents a set of rewrite rules for eliminating upward modalities, similar to some of our results in Section 5. However, [18] focuses on a single large XPath fragment, and their results do not apply to the smaller fragments considered here. Normal forms for a simple fragment of XPath are examined in [25]; for tree patterns, [1,19] present minimization algorithms, which can be viewed as a certain normal form for tree patterns.

The axiomatization of expression equivalence has been investigated for some formalisms related to XPath: [20] shows that there is no finite axiom system for regular expressions, while [11] gives axiom systems for PDL with converse. Elimination of inverse roles (upward modality) has been studied for description logics [4]. These results do not carry over to our setting since they deal with general structures (graphs) instead of trees, and they support a negation operator, thus differing radically from XPath, which (as we shall see) is not negation-closed.

**Organization**: Section 2 defines our XPath fragments, followed by a characterization of their expressive power in Section 3. Closure properties are investigated in Sections 4 under general equivalence. Section 5 revisits expressiveness and closure properties under root equivalence. Section 6 provides axiom systems and normal forms for several fragments, and Section 7 summarizes the main results of the paper.

## 2 Notations

XPath expressions are built up from an infinite set of labels (tags, names) $\Sigma$. The largest fragment of XPath studied in this paper, denoted by $\mathcal{X}^{\uparrow}_{r,[\,]}$, is syntactically defined as follows:

$$p ::= \epsilon \mid \emptyset \mid l \mid \downarrow \mid \uparrow \mid \downarrow^* \mid \uparrow^* \mid p/p \mid p \cup p \mid p[q],$$

where $\epsilon, \emptyset, l$ denote the empty path, the empty set, and a name in $\Sigma$, respectively; '$\cup$' and '$/$' stand for union and concatenation, '$\downarrow$' and '$\uparrow$' for the *child*-axis and *parent*-axis, '$\downarrow^*$' and '$\uparrow^*$' for the *descendant-or-self*-axis and *ancestor-or-self*-axis, respectively; and finally, $q$ in $p[q]$ is called a *qualifier* and defined by

$$q \quad ::= \quad p \mid \text{label} = l,$$

where $p$ is an $\mathcal{X}^{\uparrow}_{r,[\,]}$ expression and $l$ is a name in $\Sigma$.

The semantics of XPath expressions is given with respect to an XML document modeled as a node-labeled tree, referred to as an *XML tree*. Each node $n$ in an XML tree $T$ (denoted by $n \in T$) is labeled with a tag name from some finite alphabet $\Sigma_0 \subset \Sigma$ (assume, w.l.o.g., that $\Sigma_0$ has at least two symbols in it). It also has a (possibly empty) list of children; it is called a *leaf* of $T$ if it has no children. A distinguished node $rt$ is called the *root* of $T$; each node in $T$ except for the root has a parent. We do not consider order of nodes in $T$ since our XPath fragments ignore the order.

In an XML tree $T$, an $\mathcal{X}^{\uparrow}_{r,[\,]}$ expression $p$ is interpreted as a binary predicate on the nodes of $T$. That is, for any $n, n' \in T$, $T \models p(n, n')$ iff one of the following is satisfied: (1) if $p = \epsilon$, then $n = n'$; (2) if $p = \emptyset$, then $T \models p(n, n')$ is false for any $n'$; (3) if $p = l$, then $n'$ is a child of $n$, and is labeled with $l$; (4) if $p = \downarrow$, then $n'$ is a child of $n$, and its label does not matter; (5) if $p = \uparrow$, then $n'$ is the parent of $n$; (6) if $p = \downarrow^*$, then $n'$ is either $n$ itself or a descendant of $n$; (7) if $p = \uparrow^*$, then $n'$ is either $n$ itself or an ancestor of $n$; (8) if $p = p_1/p_2$, then there exists $x \in T$ such that $T \models p_1(n, x) \wedge p_2(x, n')$; (9) if $p = p_1 \cup p_2$, then $T \models p_1(n, n') \vee p_2(n, n')$; (10) if $p = p_1[q]$, then there are two cases: when $q$ is $p_2$, there exists $n'' \in T$ such that $T \models p_1(n, n') \wedge p_2(n', n'')$; when $q$ is a label test "label $= l$", $n'$ is labeled with $l$. This is in the spirit of [23].

**Example.** Referring to a node $n$ in an XML tree $T$, some $\mathcal{X}^{\uparrow}_{r,[\,]}$ expressions and their semantics are:

$p_1$: $\downarrow/A$: all the grandchildren of $n$ that are labeled $A$;

$p_2$: $\uparrow/A$: all the $A$-siblings of $n$, including $n$ itself if it is labeled $A$;

$p_3$: $\downarrow/A[\downarrow]$: all the non-leaf $A$-grandchildren of $n$;

$p_4$: $\uparrow[\text{label} = A]$: the parent of $n$ if it is labeled $A$, and the empty set otherwise;

$p_5$: $\downarrow^*/A/\downarrow^*$: all the descendants of $n$ that have an $A$-ancestor, which itself is a descendant of $n$;

$p_6$: $\downarrow^*/A[B[\downarrow^*/C]]$: all the $A$-descendants of $n$ that have a $B$-child, which in turn has a $C$-descendant;

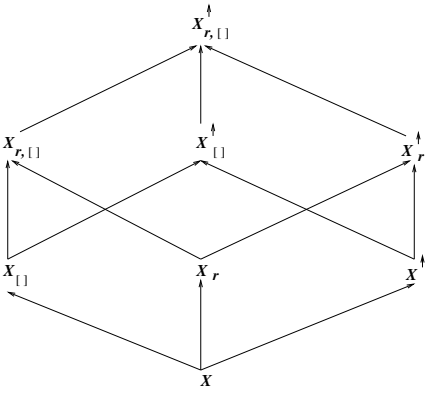


**Fig. 1.** Fragments of XPath

$p_7$: $\downarrow^*/B/\uparrow^*$: nodes having a $B$-descendant which is also a descendant of $n$; note that these nodes are not necessarily themselves descendants of $n$;

$p_8$: $\downarrow^*/A[\uparrow^*/B]$: all the $A$-descendants of $n$ that have an ancestor with a $B$-child.

□

If $T \models p(n, n')$ then $n'$ is called *reachable* from $n$ via $p$. We use $n[\![p]\!]$ to denote the set of all the nodes reached from $n$ via $p$, i.e., $\{n' | n' \in T,\ T \models p(n, n')\}$.

**Example**. For any leaf $n$ in $T$, $n[\![\downarrow]\!] = \emptyset$, $n[\![\downarrow^*]\!] = \{n\}$, while $rt[\![\uparrow]\!] = \emptyset$ and $rt[\![\uparrow^*]\!] = \{rt\}$ for the root $rt$ of $T$. □

Two XPath expressions $p$ and $p'$ are *(generally) equivalent*, denoted $p \equiv p'$, iff for any tree $T$ and node $n \in T$, $n[\![p]\!] = n[\![p']\!]$. Two expressions are *equivalent over* $\Sigma_0$ (denoted $\equiv_{\Sigma_0}$) where $\Sigma_0$ is a fixed finite alphabet, if the above holds for any tree $T$ whose labels are in $\Sigma_0$. For example, $\downarrow$ is equivalent to $A \cup B$ over the alphabet $\{A, B\}$, but not in general. We usually work with the stronger notion of general equivalence $\equiv$, and specify when results also hold for restricted equivalence — equivalence w.r.t. some finite alphabet $\Sigma_0$.

We use the following notations for subclasses of $\mathcal{X}^{\uparrow}_{r,[\,]}$: the subscript '[ ]' means that the subclass allows qualifiers, i.e., expressions of the form $p[q]$; subscript '$r$' indicates the support for "recursion" $\downarrow^*$; and superscript '$\uparrow$' denotes the support for upward modality '$\uparrow$' (and '$\uparrow^*$' in presence of the subscript $r$).

The subclasses of $\mathcal{X}^{\uparrow}_{r,[\,]}$ considered can be classified into two categories: with or without recursion. In the first category, $\mathcal{X}^{\uparrow}_{r}$ is the subclass without qualifiers; $\mathcal{X}_{r,[\,]}$ is the subclass with qualifiers but without $\uparrow$ and $\uparrow^*$; and $\mathcal{X}_r$ is the subclass with neither qualifiers, $\uparrow$ nor $\uparrow^*$. In the second category, $\mathcal{X}^{\uparrow}_{[\,]}$ is the subclass of $\mathcal{X}^{\uparrow}_{r,[\,]}$ without $\downarrow^*$ and $\uparrow^*$; $\mathcal{X}_{[\,]}$ and $\mathcal{X}^{\uparrow}$ further restrict $\mathcal{X}^{\uparrow}_{[\,]}$ by disallowing, respectively, $\uparrow$ and qualifiers; finally, $\mathcal{X}$ is the subclass of $\mathcal{X}^{\uparrow}_{[\,]}$ with neither $\uparrow$ nor qualifiers. All these fragments are useful in practice. For example, $\mathcal{X}_r$ is used by XML Schema [22] to specify integrity constraints. The proposition below justifies the lattice in Fig. 1.

**Proposition 1.** *The fragments form the lattice of Fig. 1 – there is an edge from $F_1$ to $F_2$ iff $F_1 \subseteq F_2$, i.e., if every expression of $F_1$ is equivalent to one in $F_2$.* □

**Example**. The XPath expressions given above can be classified as follows: $p_1$ is in all these fragments; $p_2$ is in $\mathcal{X}^{\uparrow}$, $\mathcal{X}^{\uparrow}_{[\,]}$, $\mathcal{X}^{\uparrow}_{r}$, $\mathcal{X}^{\uparrow}_{r,[\,]}$, but is not in the others; $p_3$ is in $\mathcal{X}_{[\,]}$, $\mathcal{X}^{\uparrow}_{[\,]}$, $\mathcal{X}_{r,[\,]}$, and $\mathcal{X}^{\uparrow}_{r,[\,]}$ only, while $p_4$ is in $\mathcal{X}^{\uparrow}_{[\,]}$ and $\mathcal{X}^{\uparrow}_{r,[\,]}$ only; $p_5$ is in $\mathcal{X}_{r}$, $\mathcal{X}^{\uparrow}_{r}$, $\mathcal{X}_{r,[\,]}$ and $\mathcal{X}^{\uparrow}_{r,[\,]}$ but is not in the others; $p_6$ is in $\mathcal{X}_{r,[\,]}$, $\mathcal{X}^{\uparrow}_{r,[\,]}$ only while $p_7$ is in $\mathcal{X}^{\uparrow}_{r}$ and $\mathcal{X}^{\uparrow}_{r,[\,]}$ only; finally, $p_8$ is only in $\mathcal{X}^{\uparrow}_{r,[\,]}$. □

A weaker equivalence relation is defined as: $p$ and $p'$ are *root equivalent*, denoted by $p \equiv_r p'$, iff for any XML tree $T$, $rt[\![p]\!] = rt[\![p']\!]$, where $rt$ is the root of $T$. Section 5 shows that root equivalence flattens the hierarchy of Fig. 1.

The fragments studied by [14] do not support upward traversals and union, and are properly contained in $\mathcal{X}_{r,[\,]}$. The distinction between root equivalence and general equivalence was not relevant to the fragments of [14], since these notions are the same in the absence of upward modalities. Although [9] considers fragments similar to $\mathcal{X}^{\uparrow}_{r,[\,]}$, it does not study the closure properties and axiom systems investigated here.

## 3 Logic and Qualified Fragments

We characterize the expressiveness of each of the fragments with qualifiers defined in the previous section, by means of both predicate logic and tree patterns. This characterization is not only interesting in its own right, but is also useful in the analysis of the closure properties of our fragments.

We start by considering disjunction and conjunction. One could extend qualifiers in an $\mathcal{X}^{\uparrow}_{r,[\,]}$ expression $p[q]$ by including conjunction and disjunction, i.e., qualifiers of the forms $q_1 \wedge q_2$ and $q_1 \vee q_2$, with the semantics: at any node $n$ in an XML tree $T$, $[q_1 \wedge q_2]$ holds iff there exist nodes $n'$ and $n''$ such that both $q_1(n, n')$ and $q_2(n, n'')$, and similarly for $[q_1 \vee q_2]$. Denote this extension of $\mathcal{X}^{\uparrow}_{r,[\,]}$ by $\mathcal{X}^{\uparrow}_{r,[\wedge,\vee]}$. This does not add expressive power over $\mathcal{X}^{\uparrow}_{r,[\,]}$, or to any of the other fragments with qualifiers.

**Proposition 2.** *$\mathcal{X}^{\uparrow}_{r,[\,]}$ and $\mathcal{X}^{\uparrow}_{r,[\wedge,\vee]}$ are equivalent.* □

We next show that the fragments with qualifiers can be captured using a version of *positive-existential first-order logic*, and define notions of tree patterns to do so. A precursor to this work is [14], which observes that the subset of $\mathcal{X}_{r,[\,]}$ consisting of expressions built up without the union operator '$\cup$' is characterized by a natural notion of "unary tree pattern".

Let $\exists^{+}(\mathit{child})$ be the fragment of first order logic built from the relation *child*, label predicates $P(x)$ for each name $P$ as well as equality '=', by closing under $\wedge$, $\vee$ and $\exists$, while $\exists^{+}(\mathit{child}, \mathit{desc})$ is the corresponding fragment built from *child*, *desc*, the label predicates and '='. The semantics is the standard semantics of first-order logic over trees (with *child* and *desc* given their standard interpretation within a tree: $\mathit{child}(x, y)$ iff $y$ is a child of $x$, and $\mathit{desc}(x, y)$ iff

$y$ is a descendant of $x$). We use $\exists^+(child)(c,s)$ to denote $\exists^+(child)$ formulae with exactly the variables $c$ and $s$ free, and similarly for $\exists^+(child, desc)(c,s)$. A $\exists^+(child)(c,s)$ formula defines a function from a node $c$ to a set of nodes $s$.

A *forest pattern* $pc$ is a forest with nodes labeled by names or the wildcard symbol '$*$' (that matches any node), and edges labeled with $\downarrow$ or $\downarrow^*$. It has a distinguished node called the *context node*, and a distinguished subset of nodes referred to as the *selected nodes* of $pc$. A pattern $pc$ thus has the form $(V, E, l, c, S)$, where $V$ is the underlying forest domain, $E$ the ordering, $l$ the labeling function, and $c$, $S$ the context node and selected set, respectively.

A forest pattern can be given semantics by translating it into $\exists^+(child, desc)$. To translate $(V, E, l, c, S)$, let $V = v_1, \ldots, v_n$ be the elements of $V$ where $v_0$ is the context node, $v_1, \ldots, v_k$ are the selected nodes, and $v_{k+1}, \ldots, v_n$ are the remaining nodes. Let $\phi(x_{v_0}, \ldots, x_{v_k})$ be the formula:

$$\exists\, x_{v_{k+1}} \ldots x_{v_n}\ (\bigwedge_{P} \bigwedge_{v \in V, l(v)=P} P(x_v)\ \wedge \bigwedge_{(v,v') \in E, l(v,v')=child} child(x_v, x_{v'}) \wedge \bigwedge_{(v,v') \in E, l(v,v')=desc} desc(x_v, x_{v'}))$$

A *tree pattern* is a forest pattern consisting of a single tree; a *child pattern* is one in which all edges are labeled with $\downarrow$; and a *unary pattern* is one in which the selected set $S$ has exactly one node.

A *forest pattern query* is a finite set of forest patterns, with all patterns having the same cardinality for the selected set $S$ (designed to model the arity of the query). A forest pattern query $t$ returns, given a tree and a distinguished context node, the union of all outputs returned by the patterns in it, i.e., $[t] = \bigcup_{tp \in t} [tp]$, where $[tp]$ is the relation defined by $tp$. By convention the empty forest pattern query is equivalent to *false*. We likewise talk about a child pattern query, unary tree pattern query, etc (note that a tree pattern query is a *finite set* of tree patterns). Let $\bigcup\mathrm{TP}(child)$ be the set of unary child tree pattern queries and $\bigcup\mathrm{TP}(child, desc)$ be the set of unary tree pattern queries.

It is easy to see that a tree pattern query can be expressed in $\exists^+(child, desc)$, and that a child pattern query can be expressed in $\exists^+(child)$. A unary pattern can be translated as above to a two-variable formula $\phi(x_{v_0}, x_{v_n})$. The following result will be used heavily in the remainder of the paper:

**Theorem 1.** *The following languages are equivalent in expressive power:*

1. $\mathcal{X}^{\uparrow}_{r,[\,]}$,
2. $\bigcup TP(child, desc)$,
3. $\exists^+(child, desc)(c,s)$. □

*Proof Sketch. 1.* $\bigcup\mathrm{TP}(child, desc) \subseteq \mathcal{X}^{\uparrow}_{r,[\,]}$. It suffices to show that each unary tree pattern has an equivalent formula in $\mathcal{X}^{\uparrow}_{r,[\,]}$. We consider the case where $c$ is neither a descendant nor an ancestor of $s$; the other two cases are similar but simpler. We first define a mapping $Q$ from nodes $v \in V$ to qualifiers. The mapping is defined inductively from the leaves:

$$Q(v) = (\mathit{label} = l(v)) \qquad \texttt{-- if } l(v) \neq \text{`}*\text{'}$$
$$\wedge \bigwedge_{(v,v')\in E,\, l(v,v')=\mathit{desc}} \downarrow^*/[Q(v')] \wedge \bigwedge_{(v,v)\in E,\, l(v,v')=\mathit{child}} \downarrow/[Q(v')]$$

Let $w$ be the first common ancestor or $c$ and $s$. The pattern is then translated to an $\mathcal{X}^{\uparrow}_{r,[\,]}$ expression that first moves upward to the node $w$, asserting at each node $v$ the qualifier $Q(v)$. At $w$, the expression asserts the existence of a path to the root node $t$ (with each node on the path qualified further by $Q(v)$). From $w$, the expression continues with a path from $w$ to the selected node $s$, again asserting the qualifier $Q(v)$ at every node $v$ in the path.

*2.* $\mathcal{X}^{\uparrow}_{r,[\,]} \subseteq \exists^+(\mathit{child}, \mathit{desc})(c, s)$. This is immediate from the definition (logic translation) of the semantics of $\mathcal{X}^{\uparrow}_{r,[\,]}$ expressions given in the previous section.

*3.* $\exists^+(\mathit{child}, \mathit{desc})(c, s) \subseteq \bigcup \mathrm{TP}(\mathit{child}, \mathit{desc})$. Let $\phi$ in $\exists^+(\mathit{child}, \mathit{desc})$ be of the form $\exists x_1 \ldots x_n \gamma$ where $\gamma$ is quantifier-free. Create a graph whose vertices are the variables of $\gamma$ and whose edges correspond to the atomic formulae in $\gamma$. Combining strongly-connected components into a single node and identifying variables that are forced by $\gamma$ to be the same yield an equivalent formula for which the corresponding graph is a tree, which gives us a tree pattern query. □

Immediately from this theorem it follows that $\mathcal{X}^{\uparrow}_{r,[\,]}$ expressions can be evaluated with a simple two-pass algorithm, taking advantage of its tree-pattern characterization.

We now consider how to characterize $\mathcal{X}^{\uparrow}_{[\,]}$. To capture the expressive power precisely, let $\exists^+(\mathit{child})[\mathit{loc}]$ be the first-order logic built up from *child* and the label predicates via conjunction, disjunction and quantification $\exists x \in B(c, n)$ (*local quantification*), for every integer $n$. Here $\exists x \in B(c, n)\ \phi(x, \boldsymbol{y})$ holds iff there is a connected set of nodes of size at most $n$ in the tree that contains $x$ and $c$. Let $\exists^+(\mathit{child})[\mathit{loc}](c, \boldsymbol{s})$ be the fragment of $\exists^+(\mathit{child})[\mathit{loc}]$ with the further restriction that each $s_i$ is restricted to be in the ball of radius $n$ around $c$.

**Theorem 2.** *The following languages are equivalent in expressive power:*

1. $\mathcal{X}^{\uparrow}_{[\,]}$,
2. $\bigcup \mathit{TP}(\mathit{child})$,
3. $\exists^+(\mathit{child})[\mathit{loc}](c, s)$. □

Theorems 1 and 2 imply that equality test in qualifiers adds no expressive power over $\mathcal{X}^{\uparrow}_{[\,]}$ and $\mathcal{X}^{\uparrow}_{r,[\,]}$. XPath allows qualifiers of the form $[q_1 = q_2]$ with the semantics: at any node $n$ in an XML tree $T$, $[q_1 = q_2]$ is true iff there exists a node $n'$ such that both $q_1(n, n')$ and $q_2(n, n')$ hold. Note that the semantics of equality test in XPath is quite different from that of qualifier conjunction.

**Corollary 1.** *The extensions of* $\mathcal{X}^{\uparrow}_{[\,]}$ *and* $\mathcal{X}^{\uparrow}_{r,[\,]}$ *to allow the equality test* $q_1 = q_2$ *in qualifiers are the same as* $\mathcal{X}^{\uparrow}_{[\,]}$ *and* $\mathcal{X}^{\uparrow}_{r,[\,]}$*, respectively.* □

*Proof Sketch.* This is because statements of the form $[q_1 = q_2]$ can clearly be expressed in $\exists^+(\mathit{child})$ and $\exists^+(\mathit{child}, \mathit{desc})$: let $\phi_1(s_1)$ and $\phi_2(s_2)$ be the $\exists^+(\mathit{child})$ (resp. $\exists^+(\mathit{child}, \mathit{desc})$) formulae representing $q_1$ and $q_2$, with free variables $s_1$ and

$s_2$ denoting the selected nodes; then $[q_1 = q_2]$ is equivalent to $\exists x(\phi_1(x) \wedge \phi_2(x))$. □

For example, $[A/\downarrow^*/B/\downarrow^* = A/\downarrow^*/C/\downarrow^*]$ can be converted to $[A/\downarrow^*/B/\downarrow^*/C/\downarrow^* \cup A/\downarrow^*/C/\downarrow^*/B/\downarrow^*]$.

There is an additional equivalence between the full logic $\exists^+(child)$ and forest patterns, which is shown in a similar way.

**Theorem 3.** *Every formula in $\exists^+(child)$ is equivalent to a child forest query, and vice versa.* □

Consider now the downward fragments. Let $\bigcup\mathrm{TP}(child, desc)[down]$ be the tree pattern queries where the node labeled $c$ is restricted to be at the root of each query. Let $\exists^+(child, desc)[down](c, s)$ be the fragment of $\exists^+(child, desc)(c, s)$ in which every bound variable as well as $s$ is syntactically restricted to be a descendant-or-self of the node $c$. Then similarly to Theorem 1:

**Theorem 4.** *The following languages are equivalent in expressive power:*

1. $\mathcal{X}_{r,[\ ]}$,
2. $\bigcup TP(child, desc)[down]$,
3. $\exists^+(child, desc)[down](c, s)$. □

Now we give the characterizations for $\mathcal{X}_{[\ ]}$. Define $\bigcup\mathrm{TP}(child)[down]$ to be the fragment of $\bigcup\mathrm{TP}(child)$ with the node $c$ at the root of each tree pattern, and let $\exists^+(child)[loc, down]$ be the intersection of the languages $\exists^+(child)[loc]$ and $\exists^+(child, desc)[down]$, i.e., every variable and also $s$ are syntactically restricted to be a fixed descendant of $c$. Then similar to Theorem 2:

**Theorem 5.** *The following languages are equivalent in expressive power:*

1. $\mathcal{X}_{[\ ]}$,
2. $\bigcup TP(child)[down]$,
3. $\exists^+(child)[loc, down](c, s)$. □

## 4 Closure Properties

An XML query often uses union, intersection and complementation of XPath expressions. This motivates the study of closure properties under these Boolean operations, i.e., whether Boolean operations preserve our fragments. This is important for, among other things, query optimization. The XPath 2.0 draft [2] proposes adding these operations; however, closure properties of our fragments remain relevant, as most applications will use only some of XPath 2.0, and implementations will focus their optimization efforts on subsets of the language.

Formally, a fragment $F$ of XPath is *closed under intersection* iff for any expressions $p_1$, $p_2$ in $F$, there exists an expression $p$ in $F$, denoted by $p_1 \cap p_2$, such that $n[\![p]\!] = n[\![p_1]\!] \cap n[\![p_2]\!]$ for any XML tree $T$ and any node $n \in T$. Similarly, $F$ is *closed under complementation* iff for any $p$ in $F$, there exists $p'$ in $F$, denoted by $\neg p$, such that $n[\![p']\!] = \{n' \mid n \in T,\ n \notin n[\![p]\!]\}$ for any XML tree $T$ and $n \in T$. Closure under union can be defined similarly; all of our fragments are closed under union since they all contain the operator '$\cup$'.

**Theorem 6.** *The fragments* $\mathcal{X}$, $\mathcal{X}_{[\,]}$, $\mathcal{X}^{\uparrow}_{[\,]}$, $\mathcal{X}_r$, $\mathcal{X}_{r,[\,]}$, *and* $\mathcal{X}^{\uparrow}_{r,[\,]}$ *are closed under intersection, whereas* $\mathcal{X}^{\uparrow}$ *and* $\mathcal{X}^{\uparrow}_r$ *are not.* □

*Proof Sketch.* Theorems 1, 2, 4 and 5 characterize the qualified fragments $\mathcal{X}^{\uparrow}_{r,[\,]}$, $\mathcal{X}^{\uparrow}_{[\,]}$, $\mathcal{X}_{r,[\,]}$ and $\mathcal{X}_{[\,]}$ with the respective logics $\exists^{+}(child, desc)(c,s)$, $\exists^{+}(child)(c,s)$, $\exists^{+}(child, desc)[down](c,s)$, and $\exists^{+}(child)[loc, down](c,s)$. In particular, for expressions $p_1$ and $p_2$ in any of these fragments, their intersection can be expressed in the corresponding logic as $\exists y_1 \phi_1(x,y_1) \wedge \exists y_2 \phi_2(x,y_2)$, where $\phi_1$ and $\phi_2$ are the codings of $p_1$ and $p_2$. Therefore the fragments are closed under intersection.

We show that $\mathcal{X}_r$ is closed under intersection using an automata-theoretic approach. A similar argument also works for $\mathcal{X}$. Let $p$ be an expression in $\mathcal{X}_r$. Acceptance of a node $n$ by such a $p$ depends only on the labels on a path from the context node $c$ to $n$, hence $p$ can be modeled by an automaton that accepts *strings* with $c$ being its start state. The automaton corresponding to $p$ can be taken to be acyclic, with the exception of self-loops labeled with the entire alphabet, and all such automata correspond to $\mathcal{X}_r$-expressions. Finally, these restricted automata are closed under the standard product construction, which accepts the intersection.

To show the negative results, consider $\epsilon \cap \uparrow/A/\uparrow/\downarrow$ – this returns the context node alone, but only if it has a sibling labeled '$A$'. We can show that this cannot be expressed in $\mathcal{X}^{\uparrow}_r$, and thus not in $\mathcal{X}^{\uparrow}$. □

**Theorem 7.** *None of the fragments is closed under complementation.* □

*Proof Sketch.* We consider here the case of equivalence over a fixed finite alphabet $\Sigma_0$ – the proofs will also work for general equivalence, but in such a case one could simply show that the complement of the simple path "$A$" cannot be represented in any of the fragments. We distinguish between two groups of fragments, those with recursion ($\downarrow^*$ and possibly $\uparrow^*$), and those without.

First, consider a non-recursive fragment. Let $p_1$ be the simple path "$A$". We claim that $p = \neg(p_1)$ cannot be represented in this fragment. To see this, let $m$ be the size of $p$. Then $p$ cannot match any path of length $> m$, a contradiction.

For recursive fragments, w.r.t. equivalence over a fixed finite alphabet $\Sigma_0$, this argument fails: if the alphabet is $\{A, A_2, \ldots, A_n\}$, $\neg A$ is $\epsilon \cup A_2 \cup \cdots \cup A_n \cup \downarrow/\downarrow/\downarrow^*$. We use a different approach, and let $p_1 = \downarrow^*/A/\downarrow^*$. Note that $p_1$ does not use of $\uparrow$ or qualifiers, and hence is in all recursive fragments. We claim that $\neg p_1$ cannot be expressed in $\mathcal{X}^{\uparrow}_{r,[\,]}$, and hence not in any of the smaller fragments.

We show this by proving that the complement of $p_1$ cannot be expressed in $\exists^{+}(child, desc)(c,s)$. Let $p$ be a representation of $\neg p_1$ in this logic, of size $m$. Consider two structures $T_1$ and $T_2$. The structure $T_1$ consists of a root $rt$ followed by a chain of length $2^{m+1}$, with all these nodes labeled $B$ (and hence the end of the chain is in the complement with respect to context node $rt$). The structure $T_2$ is similar, except that a node in the middle of the chain is labeled $A$. Then every $m$-quantifier $\exists^{+}(child, desc)(c,s)$ sentence holding in $T_1$ also holds in $T_2$, a contradiction. □

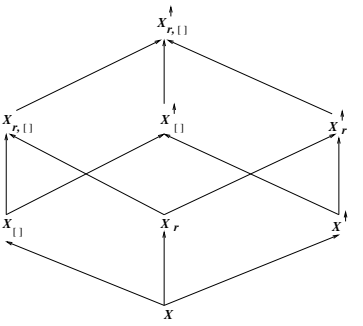


**Fig. 2.** XPath fragments under root equivalence

## 5 Root Equivalence

Recall the notion of root equivalence defined in Section 2. Under this weaker equivalence relation, the hierarchy of Fig. 1 collapses: the XPath fragments form the lattice of Fig. 2. This follows from:

**Theorem 8.** *Under root equivalence, (1)* $\mathcal{X}_r^{\uparrow} \subset \mathcal{X}_{r,[\,]}$ *but* $\mathcal{X}_{r,[\,]} \not\subseteq \mathcal{X}_r^{\uparrow}$*; (2)* $\mathcal{X}^{\uparrow} \subseteq \mathcal{X}_{[\,]}$*; moreover, in the absence of label tests, i.e., qualifiers of the form* $[label = l]$*,* $\mathcal{X}^{\uparrow} = \mathcal{X}_{[\,]}$*; (3)* $\mathcal{X}_{r,[\,]} = \mathcal{X}_{r,[\,]}^{\uparrow}$*; and (4)* $\mathcal{X}_{[\,]} = \mathcal{X}_{[\,]}^{\uparrow}$*.* □

As an immediate consequence, any XPath expression with upward modalities in any of these fragments is root-equivalent to an XPath expression with neither $\uparrow$ nor $\uparrow^*$. This is important for, among other things, processing streaming data.

*Proof Sketch. 1.* $\mathcal{X}_r^{\uparrow} \subset \mathcal{X}_{r,[\,]}$. Note that $p/(p_1 \cup p_2)/p' \equiv_r p/p_1/p' \cup p/p_2/p'$. Thus, w.l.o.g., assume that any $\mathcal{X}_r^{\uparrow}$ expression is of the form $p_1 \cup \cdots \cup p_k$, where each $p_i$ $(1 \leq i \leq k)$ has the form $\eta_1/\cdots/\eta_m$, and each $\eta_j$ is one of $\emptyset$, $\epsilon$, $l$, $\downarrow$, $\downarrow^*$, $\uparrow$, or $\uparrow^*$. It suffices to show that each $p_i$ can be converted to an equivalent $\mathcal{X}_{r,[\,]}$ expression. The conversion is from left to right, starting with the first occurrence of $\uparrow$ or $\uparrow^*$, using the rewriting rules (omitting trivial rules for $\emptyset$ and $\epsilon$):

$$\begin{aligned}
\uparrow/p &\Rightarrow \emptyset, \text{ if } \uparrow \text{ is the first symbol of } p_i\,;\\
\uparrow^*/p &\Rightarrow p, \text{ if } \uparrow^* \text{ is the first symbol of } p_i\,;\\
p/\eta/\uparrow &\Rightarrow p[\eta], \text{ where } \eta \text{ is } \downarrow \text{ or a label}\,;\\
\eta_1/\cdots/\eta_n/\uparrow^* &\Rightarrow \eta_1[\eta_2/\cdots/\eta_n] \cup \cdots \cup \eta_1/\cdots/\eta_i[\eta_{i+1}/\cdots/\eta_n] \cup \cdots\\
&\qquad \cup\, \eta_1/\cdots/\eta_n, \text{where } \eta_i \text{ is } \downarrow, \downarrow^* \text{ or a label.}
\end{aligned}$$

In other words, if $\eta_1$ is $\uparrow$ (resp. $\uparrow^*$), the first (resp. second) rule is applied to eliminate it; otherwise eliminate $\uparrow$ and $\uparrow^*$ by introducing qualifiers using the other rules. In these rules, $p$, $p_1$, and $p_2$ denote $\mathcal{X}_r^{\uparrow}$ expressions without $\uparrow$, $\uparrow^*$, and the left-to-right conversion implies that a rewriting rule is applied only if its left-hand side matches the prefix of an $\mathcal{X}_r^{\uparrow}$ expression. Although this might introduce '$\cup$', one needs only to consider $p_i$'s without '$\cup$' because of the rewriting

rules for '$\cup$'. Each rewriting rule is root-equivalence preserving and $p_i$ can be converted to an $\mathcal{X}_{r,[\,]}$ expression in a finite number of steps. The containment is proper because the $\mathcal{X}_{r,[\,]}$ expression $\downarrow^*/A[\downarrow^*/B]$ is not root-equivalent to any $\mathcal{X}_r^{\uparrow}$ expression.

*2.* $\mathcal{X}^{\uparrow} \subseteq \mathcal{X}_{[\,]}$, and furthermore, $\mathcal{X}^{\uparrow} = \mathcal{X}_{[\,]}$ in the absence of label tests. The proof that $\mathcal{X}_r^{\uparrow} \subseteq \mathcal{X}_{r,[\,]}$ given above also shows $\mathcal{X}^{\uparrow} \subseteq \mathcal{X}_{[\,]}$. For the other direction, it suffices to show that without label tests in qualifiers, each $\mathcal{X}_{[\,]}$ expression can be rewritten to an equivalent $\mathcal{X}_r^{\uparrow}$ expression. This is done by eliminating the qualifiers using the rewriting rules, similar to part *1.*

Note that in the presence of label tests, $\mathcal{X}_{[\,]} \not\subseteq \mathcal{X}^{\uparrow}$. A concrete example of this is $\epsilon[label = A]$, i.e., testing whether the root is labeled $A$. It is easy to show that this $\mathcal{X}_{[\,]}$ expression is not expressible in $\mathcal{X}^{\uparrow}$.

*3.* $\mathcal{X}_{r,[\,]} = \mathcal{X}_{r,[\,]}^{\uparrow}$. From Theorems 1 and 4, $\mathcal{X}_{r,[\,]}$ and $\mathcal{X}_{r,[\,]}^{\uparrow}$ are equivalent to the tree pattern queries $\bigcup\mathrm{TP}(child, desc)$ and $\bigcup\mathrm{TP}(child, desc)[down]$. Recall that $\bigcup\mathrm{TP}(child, desc)[down]$ is defined to be the fragment of $\bigcup\mathrm{TP}(child, desc)$ where the context node labeled $c$ is restricted to be at the root of each query. Under root equivalence, the context node $c$ is explicitly restricted to be at the root, and so $\bigcup\mathrm{TP}(child, desc)$ and $\bigcup\mathrm{TP}(child, desc)[down]$ have the same expressive power under root equivalence; the same holds for $\mathcal{X}_{r,[\,]}^{\uparrow}$ and $\mathcal{X}_{r,[\,]}$.

*4.* $\mathcal{X}_{[\,]} = \mathcal{X}_{[\,]}^{\uparrow}$. In a similar way to part *3*, Theorems 2 and 5 show that $\mathcal{X}_{[\,]}$ and $\mathcal{X}_{[\,]}^{\uparrow}$ are root equivalent to each other. □

The following can now be easily verified along the same lines as Proposition 1.

**Corollary 2.** *Under root equivalence, the fragments form the lattice of Fig. 2; that is, there is an edge from fragment $F_1$ to $F_2$ iff $F_1 \subseteq F_2$, i.e., if every expression of $F_1$ is root-equivalent to an expression in $F_2$.* □

Recall that $\mathcal{X}^{\uparrow}$ is not closed under intersection as far as general equivalence is concerned. For root equivalence, the situation is different: Theorems 6, 7 and 8 imply:

**Corollary 3.** *Under root equivalence, all fragments except $\mathcal{X}_r^{\uparrow}$ are closed under intersection but remain not closed under complementation.* □

## 6 Axiom Systems and Normal Forms

We next study axiomatizability and normal forms for XPath, giving preliminary results for the fragments $\mathcal{X}_r$ and $\mathcal{X}$. Although the results of this section are established under full equivalence, they also hold under root equivalence.

### 6.1 Axiom Systems

An XPath term over $F$ is built up from the constructors of $F$, but supplementing the labels (names) and $\epsilon$ with a set of *expression variables* (ranged over by

$p_1, \ldots, p_n$). For a term $\tau$ over $F$, the variables used in constructing $\tau$ are called the *free variables of* $\tau$. All the XPath expressions seen before are terms with no free variables, and will also be referred to as *ground terms*, or simply *expressions*.

An *equation* for a fragment $F$ is an equality of terms. Given a set of equations $A$ in $F$, the equivalence relation $\equiv_A$, is the smallest equivalence relation that contains the symmetric closure of $A$ and closed under:

1. if $\tau \equiv_A \tau'$ then $\sigma(\tau) \equiv_A \sigma(\tau')$ for any substitution $\sigma$ that maps the free variables in $\tau$ or $\tau'$ to expressions of $F$;
2. if $\tau \equiv_A \tau'$ then $\gamma \equiv_A \gamma'$, where $\gamma'$ is obtained from $\gamma$ by replacing some occurrence of subexpression $\tau$ in $\gamma$ with $\tau'$.

A set of equations $A$ is *sound* for a fragment $F$ if for all $\tau_1$ and $\tau_2$ in $F$, $\tau_1 \equiv_A \tau_2$ implies $\tau_1 \equiv \tau_2$, and is *complete* if for all such $\tau_1$ and $\tau_2$, $\tau_1 \equiv \tau_2$ implies $\tau_1 \equiv_A \tau_2$. A *(resp. finite) axiom system* for $F$ is a (resp. finite) set of equations that is sound and complete. A fragment of XPath is said to be *axiomatizable* iff it has an axiom system, and *finitely axiomatizable* iff it has a finite axiom system.

A (finite) axiom system for a fragment of XPath is useful in, among other things, optimizing and normalizing the XPath expressions. Unfortunately, none of the fragments considered in this paper is finitely axiomatizable.

**Proposition 3.** *None of the fragments is finitely axiomatizable.* □

*Proof Sketch.* We show that $\mathcal{X}$ is not finitely axiomatizable. The same proof also applies to other fragments.

Assume $A$ is a finite axiomatization for $\mathcal{X}$. Then only finitely many labels of $\Sigma$ are mentioned in $A$. Let $c$ be a name that does not appear in $A$. Note that $c \cup \downarrow \equiv \downarrow$ must hold. Since $A$ is complete, there is a proof $S$ of $c \cup \downarrow \equiv_A \downarrow$ using the inference rules and axioms in $A$. Since $c$ is not in these axioms, $c$ must be introduced by substituting some $\mathcal{X}$ expression $E$ containing $c$ for some variable $p$ using the first inference rule above. If we substitute $c/c$ for each occurrence of $c$, we get a proof $S'$ which is the same as $S$, except that $c/c$ appears in $S'$ wherever $c$ appears in $S$. Thus it is a proof for $c/c \cup \downarrow \equiv_A \downarrow$, which is false. □

### 6.2 Axiom Systems for $\mathcal{X}_r$ and $\mathcal{X}$

We next present a natural set of axiom schemas for two fragments, namely, $\mathcal{X}_r$ and $\mathcal{X}$, and show that when equivalence over a finite alphabet $\Sigma_0$ is considered, $\mathcal{X}$ is finitely axiomatizable. One application of these axiom systems, deriving a normal form for expressions in these fragments, will be shown later.

Since the equivalence problem for XPath expressions in all of our fragments is decidable (e.g., by appealing to [15] ), a trivial computable axiomatization can be taken for any of the fragments by simply enumerating all ground equalities. Such an axiomatization would not assist simplification of XPath, and would also not help in providing finite axiomatizations for restricted equivalence.

**Fragment $\mathcal{X}_r$.** Table 1 provides an axiom system for $\mathcal{X}_r$, denoted by $\mathcal{I}_r$. The system is infinite since for each label $l$ in the alphabet $\Sigma$, there is an $l$-child rule in

**Table 1.** $\mathcal{I}_r$: axioms for $\mathcal{X}_r$

| | |
|---|---|
| $\epsilon/p \equiv p \equiv p/\epsilon$ | (empty-path) |
| $\emptyset \cup p \equiv p$ | (empty-set-union) |
| $p_1/\emptyset/p_2 \equiv \emptyset$ | (empty-set-concat) |
| $l \cup \downarrow \equiv \downarrow$ | ($l$-child) |
| $\downarrow^* \equiv \epsilon \cup \downarrow/\downarrow^*$ | (descendants) |
| $p \cup \downarrow^* \equiv \downarrow^*$ | (descendants-union) |
| $\downarrow/\downarrow^* \equiv \downarrow^*/\downarrow$ | (child-descendants) |
| $p_1/(p_2/p_3) \equiv (p_1/p_2)/p_3$ | (concat-associativity) |
| $p_1 \cup (p_2 \cup p_3) \equiv (p_1 \cup p_2) \cup p_3$ | (union-associativity) |
| $p_1 \cup p_2 \equiv p_2 \cup p_1$ | (union-commutativity) |
| $p \cup p \equiv p$ | (union-idempotent) |
| $p/(p_1 \cup p_2) \equiv p/p_1 \cup p/p_2$ | (left-distributivity) |
| $(p_1 \cup p_2)/p \equiv p_1/p \cup p_2/p$ | (right-distributivity) |

$\mathcal{I}_r$. When considered with respect to equivalence over any fixed finite alphabet $\Sigma_0$, $\mathcal{I}_r$ is still sound, but it is no longer complete. Note that by the *concat-associativity* axiom in $\mathcal{I}_r$, we can write $\tau_1/\tau_2/\tau_3$ for $(\tau_1/\tau_2)/\tau_3$ and $\tau_1/(\tau_2/\tau_3)$.

**Theorem 9.** *For any $\mathcal{X}_r$ expressions $\tau$ and $\tau'$, $\tau \equiv \tau'$ iff $\tau \equiv_{\mathcal{I}_r} \tau'$.* □

**Example**. Using $\mathcal{I}_r$, $\downarrow^*/\downarrow^* \equiv \downarrow^*$ can be verified: $\downarrow^*/\downarrow^* \equiv_{\mathcal{I}_r} \downarrow^*/(\downarrow^* \cup \epsilon) \equiv_{\mathcal{I}_r} \downarrow^*/\downarrow^* \cup \downarrow^* \equiv_{\mathcal{I}_r} \downarrow^*$ by the *descendants*, *left-distribution*, and *descendants-union* axiom, respectively. □

To prove Theorem 9, we define a containment relation. An XPath expression $p_1$ is *contained* in $p_2$ (written $p_1 \subseteq p_2$) iff for any XML tree $T$ and any node $n$ in $T$, $n[\![p_1]\!] \subseteq n[\![p_2]\!]$. If containment holds only for trees over a fixed alphabet $\Sigma_0$, we write $p_1 \subseteq_{\Sigma_0} p_2$. Note that $\mathcal{I}_r$ also suffices to determine containment of $\mathcal{X}_r$ expressions:

**Lemma 1.** *For any $\mathcal{X}_r$ expressions $\tau_1$ and $\tau_2$, $\tau_1 \subseteq \tau_2$ iff $\tau_2 \cup \tau_1 \equiv_{\mathcal{I}_r} \tau_2$.* □

*Proof Sketch.* ($\Rightarrow$): The implication is proved first by assuming that $\tau_1$ is union-free (see Section 6.3 for a definition); this case can be shown by induction on the length of $\tau_1$. The general case can be reduced to the union-free case by induction on the number of union-terms. ($\Leftarrow$): This direction can be verified by induction on the length of the $\mathcal{I}_r$ proof. □

This lemma implies Theorem 9, and is also interesting in its own right, since XML queries often check containment of XPath expressions.

**Fragment $\mathcal{X}$.** We now consider $\mathcal{X}$. Let $\mathcal{I}$ be $\mathcal{I}_r$ without the axioms for *descendants*, *descendants-union* and *child-descendants*. For a finite alphabet $\Sigma_0$, let it

be enumerated as $l_1, \ldots, l_n$. A finite axiom system $\mathcal{I}^f(\Sigma_0)$ consists of rules in $\mathcal{I}$ except for the $l$-child rules and with the addition of the rule:

$$l_1 \cup \cdots \cup l_n \equiv \downarrow \qquad \text{(finite-alphabet)}$$

Note that for each $l \in \Sigma_0$, $l \cup \downarrow \equiv \downarrow$ can be derived from the *finite-alphabet* and *union-idempotent* axioms of $\mathcal{I}^f(\Sigma_0)$. One can verify that $\mathcal{I}$ is an axiomatization of $\mathcal{X}$ under general equivalence (the default notion), and that for each finite $\Sigma_0$, $\mathcal{I}^f(\Sigma_0)$ is a finite axiomatization of $\mathcal{X}$ for equivalence over $\Sigma_0$.

**Theorem 10.** *For any $\mathcal{X}$ expressions $\tau_1$ and $\tau_2$, (1) $\tau_1 \equiv \tau_2$ iff $\tau_1 \equiv_{\mathcal{I}} \tau_2$, and $\tau_1 \subseteq \tau_2$ iff $\tau_2 \cup \tau_1 \equiv_{\mathcal{I}} \tau_2$; and (2) For each finite $\Sigma_0$, $\tau_1 \equiv_{\Sigma_0} \tau_2$ iff $\tau_1 \equiv_{\mathcal{I}^f(\Sigma_0)} \tau_2$, and $\tau_1 \subseteq_{\Sigma_0} \tau_2$ iff $\tau_2 \cup \tau_1 \equiv_{\mathcal{I}^f(\Sigma_0)} \tau_2$.* □

### 6.3 Normal Forms for $\mathcal{X}_r$ and $\mathcal{X}$

We next study normal forms for $\mathcal{X}_r$ and $\mathcal{X}$. A fragment $F$ of XPath has a *normal form* if there is a function $\lambda$ from $F$ to a subset $F_{nl}$ of $F$ such that for any expressions $\tau_1$ and $\tau_2$ of $F$, $\tau_1 \equiv \tau_2$ iff $\lambda(\tau_1) = \lambda(\tau_2)$, i.e., they are syntactically equal. We shall say that $F_{nl}$ is a *normal form* of $F$. Note that $\lambda$ is determined by $F_{nl}$. For an expression $\tau$ in $F$, $\lambda(\tau)$ is called the *normal form* of $\tau$ w.r.t. $F_{nl}$, or simply the normal form if $F_{nl}$ is clear from the context. Similarly, one can define for $F$ a *normal form for equivalence* over a finite alphabet $\Sigma_0$. A normal form is useful for simplifying the analysis of equivalence since it reduces semantic equivalence to syntactic equality.

**Fragment $\mathcal{X}_r$.** An $\mathcal{X}_r$ expression is *union-free* if it does not contain union '$\cup$'. We give a normal form for union-free $\mathcal{X}_r$ expressions; normal forms for $\mathcal{X}_r$ expressions with unions are still under investigation. A union-free $\mathcal{X}_r$ expression $\alpha$ is in *normal form* if (1) it does not contain $\epsilon$ (resp. $\emptyset$) unless $\alpha = \epsilon$ (resp. $\alpha = \emptyset$); (2) any contiguous sequence in $\alpha$ consisting of $\downarrow$'s and at least one $\downarrow^*$ is of the form $\downarrow^*/\downarrow/\downarrow^*/\cdots/\downarrow/\downarrow^*$ (with the same number of occurrences of $\downarrow$); and (3) $\alpha$ does not contain consecutive $\downarrow^*/\downarrow^*$.

**Proposition 4.** *For any union-free $\mathcal{X}_r$ expression $\alpha$ there is a unique $\mathcal{X}_r$ expression $\alpha'$ such that $\alpha \equiv \alpha'$ and $\alpha'$ is in the normal form above, and, in the case of a finite alphabet $\Sigma_0$, for any union-free $\alpha$ there is such a unique $\alpha'$ such that $\alpha \equiv_{\Sigma_0} \alpha'$.* □

**Fragment $\mathcal{X}$.** We next define a normal form for general $\mathcal{X}$ expressions. An $\mathcal{X}$ expression $\tau$ is in the *union normal form* if it is of the form $\alpha_1 \cup \cdots \cup \alpha_k$, where for each $\alpha_i$, referred to as a *disjunct* of $\tau$, (1) $\alpha_i$ does not contain $\epsilon$ unless $\alpha_i = \epsilon$, and $\tau$ does not contain $\emptyset$ unless $\tau = \emptyset$; (2) $\alpha_i$ is a *union-free* expression; and (3) $\alpha_i$ is not contained in any other $\alpha_j$, i.e., $\alpha_i \not\subseteq \alpha_j$ for all $j \neq i$. When considering a fixed finite alphabet $\Sigma_0 = \{l_1, \ldots, l_n\}$, add the condition (4) $\tau$ does not contain $\downarrow$, which is represented instead by $l_1 \cup \cdots \cup l_n$. Then:

**Proposition 5.** *For any $\mathcal{X}$ expression $\tau$ there is an $\mathcal{X}$ expression $\tau'$ such that $\tau \equiv \tau'$ and $\tau'$ is in the union normal form satisfying conditions (1)–(3) given*

| | $\mathcal{X}_{[\,]}$ | $\mathcal{X}^{\uparrow}_{[\,]}$ | $\mathcal{X}_{r,[\,]}$ | $\mathcal{X}^{\uparrow}_{r,[\,]}$ |
|---|---|---|---|---|
| logic | $\exists^{+}(child)_{[loc,down](c,s)}$ | $\exists^{+}(child)_{[loc](c,s)}$ | $\exists^{+}(child,desc)_{[down](c,s)}$ | $\exists^{+}(child,desc)_{(c,s)}$ |
| pattern | $\bigcup \mathrm{TP}(child)_{[down]}$ | $\bigcup \mathrm{TP}(child)$ | $\bigcup \mathrm{TP}(child,desc)_{[down]}$ | $\bigcup \mathrm{TP}(child,desc)$ |

**Fig. 3.** Characterizations of the expressiveness of the fragments

| closure properties | | $\mathcal{X}$ | $\mathcal{X}_{[\,]}$ | $\mathcal{X}^{\uparrow}$ | $\mathcal{X}_r$ | $\mathcal{X}^{\uparrow}_{[\,]}$ | $\mathcal{X}_{r,[\,]}$ | $\mathcal{X}^{\uparrow}_r$ | $\mathcal{X}^{\uparrow}_{r,[\,]}$ |
|---|---|---|---|---|---|---|---|---|---|
| intersection | under $\equiv$ | Yes | Yes | No | Yes | Yes | Yes | No | Yes |
| | under $\equiv_r$ | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes |
| complementation | under $\equiv$ | No | No | No | No | No | No | No | No |
| | under $\equiv_r$ | No | No | No | No | No | No | No | No |

**Fig. 4.** The closure properties of the fragments

*above. For every finite alphabet $\Sigma_0$ there is $\tau' \equiv_{\Sigma_0} \tau$ with $\tau'$ satisfying (1)–(4). Moreover, in both cases $\tau'$ is unique up to ordering of the disjuncts.* □

## 7 Conclusion

We have investigated important structural properties of a variety of XPath fragments, parameterized with modalities that include qualifiers, upward traversal (the parent and ancestor axes), and recursion (descendant and ancestor). We have provided characterizations of the fragments with qualifiers in terms of both logics and tree patterns (Fig. 3), and have given a complete picture of the closure properties of all these fragments (Fig. 4), under both general equivalence and root equivalence. We have also established preliminary results on axiom systems and normal forms for some of these fragments. These results advance our understanding of different XPath modalities, and are also useful for simplification of XPath expressions and optimization of XML queries.

One open problem is the finite axiomatizability of $\mathcal{X}_r$ for $\equiv_{\Sigma_0}$, and similarly for the other fragments. Regular expressions do not have a finite axiom system when the inference rules are restricted to the ones given in Section 6, even when the alphabet consists of a single symbol [20]. Although $\mathcal{X}_r$ is a large class of regular expressions, we speculate that $\equiv_{\Sigma_0}$ is finitely axiomatizable for $\mathcal{X}_r$, and are currently investigating this issue for $\mathcal{X}_r$ terms, which may contain free variables. Another topic for future work is the expressiveness and closure properties of other fragments, especially those with negations in qualifiers. A third topic is to strengthen our logical characterizations to the case where trees have *data values*, and to make the characterizations more precise by mapping to positive existential logic with two variables. Another topic is to reinvestigate these issues in the presence of DTDs/XML Schema and integrity constraints. We are also exploring the use of our results in developing rewrite systems and algorithms for simplifying XPath expressions.

## References


1. S. Amer-Yahia, S. Cho, V. Lakshmaman, and D. Srivistava. Minimization of tree pattern queries. In *SIGMOD*, 2001.
2. A. Berglund et al. *XML Path Language (XPath) 2.0.* W3C Working Draft, Dec. 2001. `http://www.w3.org/TR/xpath20`.
3. G. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.
4. D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier, 2001.
5. D. Chamberlin et al. XQuery 1.0: An XML query language. W3C Working Draft, June 2001. `http://www.w3.org/TR/xquery`.
6. C. Chan, P. Felber, M. Garofalakis, and R. Rastogu. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.
7. J. Clark. *XSL Transformations (XSLT).* W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xslt`.
8. J. Clark and S. DeRose. *XML Path Language (XPath).* W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xpath`.
9. A. Deutsch and V. Tannen. Containment for classes of XPath expressions under integrity constraints. In *KRDB*, 2001.
10. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB*, 2002.
11. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* The MIT Press, 2000.
12. C. Hoffmann and M. O'Donnell. Pattern matching in trees. *JACM*, 29(1):68–95, 1982.
13. P. Kilpelainen and H. Manilla. Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995.
14. G. Miklau and D. Suciu. Containment and equivalence of XPath expressions. In *PODS*, 2002.
15. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS*, 2001.
16. M. Murata. Extended path expressions for XML. In *PODS*, 2001.
17. F. Neven and T. Schwentick. Expressive and efficient languages for tree-structured data. In *PODS*, 2000.
18. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *Workshop on XML-Based Data Management (XMLDM)*, 2002.
19. P. Ramanan. Efficient algorithms for minimizing tree pattern queries. In *SIGMOD*, 2002.
20. V. Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964. (Russian).
21. J. Robie, J. Lapp, and D. Schach. *XML Query Language (XQL).* Workshop on XML Query Languages, Dec. 1998.
22. H. Thompson et al. XML Schema. W3C Working Draft, May 2001. `http://www.w3.org/XML/Schema`.
23. P. Wadler. Two semantics for XPath. Technical report, Bell Labs, 2000.
24. P. Wood. On the equivalence of XML patterns. In *Computational Logic*, 2000.
25. P. Wood. Minimising simple XPath expressions. In *WebDB*, 2001.

# On Reasoning about Structural Equality in XML: A Description Logic Approach

David Toman and Grant Weddell

School of Computer Science, University of Waterloo, Canada
{david,gweddell}@uwaterloo.ca

**Abstract.** We define a boolean complete description logic dialect called $\mathcal{DLFD}_{\text{reg}}$ that can be used to reason about structural equality in semistructured ordered data in the presence of document type definitions. This application depends on the novel ability of $\mathcal{DLFD}_{\text{reg}}$ to express functional dependencies over sets of possibly infinite feature paths defined by regular languages. We also present a decision procedure for the associated logical implication problem. The procedure underlies a mapping of such problems to satisfiability problems of $\text{Datalog}_{nS}^{\vee,\neg}$ and in turn to the Ackermann case of the decision problem.

## 1 Introduction

Equality is a fundamental notion in any interaction with data, and the need to reason about equality during query optimization and evaluation is unavoidable in any practical data model and query language. Although the problem of reasoning about equality has been studied extensively for relational and object oriented data models, this is not the case for the more recent semistructured ordered models and query languages such as XML and XQuery. With XML in particular, there are three notions of equality that have surfaced in the literature. *Label* equality, based on equality of strings denoting *element tags*, and *node* equality, based on node identity, are two of these notions that have simple and efficient implementations. *Structural equality* between arbitrary ordered forests representing XML documents is a third and much more costly variant of equality. Structural equality, however, is the basis for comparing XML values in the XQuery language [5]. In particular, structural equality is heavily used by `where` clauses in FLWR expressions, by *duplicate elimination* operators, etc.

**Example 1** Consider the following XQuery expression that constructs the names of employees who have received mail:

```
for x in doc(personnel)//emp,
    y in doc(shipping)//received/emp
where x=y
return x/name
```

The detection of matching *employee* subdocuments in the `where` clause requires, according to the XQuery specification, a potentially expensive structural equality

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 96–110, 2003.

comparison [5]. However, knowing that employee documents are structurally equal if they have the same employee number (a fixed, and presumably small part of the *employee* subdocument) would enable this test to be replaced by an efficient test for node equality on respective employee number subdocuments.

This optimization depends on our ability to specify and reason about equality and, in particular, about an extended notion of *functional dependencies* that hold in XML documents. Note that detection of duplicate employee subdocuments and several other XQuery operations based on structural equality can similarly benefit from these additional reasoning capabilities.

In this paper, we propose a new approach to reasoning about structural equality in XML documents. The proposed approach is indirect; we begin by defining a boolean complete dialect of a *description logic*, $\mathcal{DLFD}_{\text{reg}}$. This dialect has the crucial ability to reason about implied structural equality using *regular path functional dependencies*. The contributions of this paper are as follows:

- We introduce the notions of *regular restrictions* and of *regular path functional dependencies*, based on regular sets of feature path descriptions, in the context of a boolean complete description logic;
- We provide a sound and complete reasoning procedure for $\mathcal{DLFD}_{\text{reg}}$, including tight complexity bounds for the associated implication problem; and
- We show a link between reasoning in this logic and reasoning about structural equality in XML documents.

The ability to formulate implication problems using path functional dependencies has a number of important applications in query optimization. In particular, the correctness of a number of query rewrite rules that relate to duplicate elimination [23], to sort operator elimination [31], and to a variety of other optimizations [29,33] can be usefully abstracted as logical implication problems. Similar results can be obtained in the XML/XQuery setting using the regular path functional dependencies proposed in this paper.

### 1.1 Related Work and Outline

There is a growing body of work on integrity constraints that have been proposed in the literature to restrict the structure of XML documents. In particular, constraints that resemble a form of keys and functional constraints have been considered in [2,3,8,17]. However, there still remains the problem of reasoning about arbitrary structural equality between (sub)documents with unstructured components.

As stated above, this issue of structural equality is the focus of this paper and is considered in the context of the description logic $\mathcal{DLFD}_{\text{reg}}$. Of particular significance is that our results compliment earlier work by Calvanese et al. [9] that was first to propose the use of a DL to reason about *document type definitions* (or DTDs). In particular, they considered the problem of comparing DTDs to determine various inclusion relationships.

In [23,24] we considered a very simple DL dialect that included an "fd" concept constructor for capturing path functional dependencies [7,33]. The implication problem for this dialect was complete for DEXPTIME despite the fact that it did not allow defined concepts in terminologies. This result was obtained by comparing the problem to the recognition problem for $\text{Datalog}_{nS}$.

The remainder of the paper is organized as follows. Section 2 defines the syntax and semantics for $\mathcal{DLFR}$, and introduces $\text{Datalog}_{nS}^{\vee,\neg}$. Section 3 studies the complexity associated with reasoning in $\mathcal{DLFD}_{\text{reg}}$; Subsection 3.1 shows DEXPTIME hardness for a fragment of $\mathcal{DLFD}_{\text{reg}}$, then Subsections 3.2 and 3.3 present DEXPTIME decision procedure for $\mathcal{DLFD}_{\text{reg}}$. Section 4 outlines how $\mathcal{DLFD}_{\text{reg}}$ can be used to reason about structural equality in XML. We conclude with a summary and a list of conjectures and open questions in Section 5.

## 2 Definitions

Apart from *concepts*, which are descriptions of sets, description logics that derive from $\mathcal{ALC}$ [30], such as $\mathcal{SHIQ}$ [20] and $\mathcal{DLR}$ [11], start with descriptions of relations as basic building blocks. Binary relations are a particular focus and are called *roles*. The description logic $\mathcal{DLFD}_{\text{reg}}$ introduced in this section takes an alternative approach that starts by augmenting concepts with the notion of *attributes* (or *features*). Attributes correspond to unary functions on an underlying object domain. This approach has several advantages. First, the decidability and complexity of logical implication for $\mathcal{DLFD}_{\text{reg}}$ is closely related to reasoning in $\text{Datalog}_{nS}^{\vee,\neg}$, a *logic programming* language of monadic predicates and functions [12,13]. Second, the use of $\text{Datalog}_{nS}^{\vee,\neg}$ provides a uniform framework in which we can study various extensions of $\mathcal{DLFD}_{\text{reg}}$, how roles can be simulated by attributes for example. And third, the connection to logic programming provides a rich set of *efficient implementation* techniques [14,16].

The syntax and semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$ is given by the following.

**Definition 2 (Regular Path Expression Sets)** *Let $F$ be a set of attributes. A* regular path expression *is defined by the grammar*

$$L ::= f \mid \mathsf{Id} \mid L_1.L_2 \mid L_1, L_2 \mid (L)*,$$

*where $f \in F$ and where* Id *denotes the empty string. A string generated by a regular path expression $L$ is called a* path expression*, and is assumed to conform to the grammar "*$\mathsf{Pf} ::= f.\mathsf{Pf} \mid \mathsf{Id}$*". The set of path expressions generated by $L$ is denoted $\mathcal{L}(L)$.*

**Definition 3 (Syntax and Semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$)** *Let $C$ be a set of primitive concepts disjoint from $F$. The derived* concept descriptions *for $\mathcal{DLFD}_{\text{reg}}$ are defined by the grammar in Figure 3. A concept formed by an application of the final production in the grammar is* equality generating*. The derived concept descriptions for $\mathcal{DLF}_{\text{reg}}$ are those in $\mathcal{DLFD}_{\text{reg}}$ that are not equality generating.*

| Syntax | | Semantics: Defn of "$(\cdot)^{\mathcal{I}}$" |
|---|---|---|
| $D$ | $::= C$ | $(C)^{\mathcal{I}} \subset \Delta$ |
| | $\mid D_1 \sqcap D_2$ | $(D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}}$ |
| | $\mid \neg D$ | $\Delta \setminus (D)^{\mathcal{I}}$ |
| | $\mid \forall L.D$ | $\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for all $\mathsf{Pf} \in \mathcal{L}(L)\}$ |
| | $\mid \exists L.D$ | $\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for some $\mathsf{Pf} \in \mathcal{L}(L)\}$ |
| $E$ | $::= D$ | |
| | $\mid D : L \rightarrow L'$ | $\Big\{x \in \Delta : \forall y \in (D)^{\mathcal{I}}. \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L)} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big) \Rightarrow \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L')} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big)\Big\}$ |

**Fig. 1.** Syntax and Semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$.

*An* inclusion dependency $\mathcal{C}$ *is an expression of the form* $D \sqsubseteq E$.

*The* semantics *of expressions is given with respect to a structure* $(\Delta, \cdot^{\mathcal{I}})$*, where* $\Delta$ *is a domain of "objects", and* $(.)^{\mathcal{I}}$ *an interpretation function that fixes the interpretations of primitive concepts to be subsets of* $\Delta$ *and attributes to be total functions over* $\Delta$*. The interpretation is extended to path expressions,* $(\mathsf{Id})^{\mathcal{I}} = \lambda x.x$ *and* $(f.\,\mathsf{Pf})^{\mathcal{I}} = (\mathsf{Pf})^{\mathcal{I}} \circ (f)^{\mathcal{I}}$*, and to derived concept descriptions, cf. Figure 3.*

*An interpretation* satisfies an inclusion dependency $\mathcal{C} (\equiv D \sqsubseteq E)$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$.

*A* terminology $\mathcal{T}$ *consists of a finite set of inclusion dependencies* $\{\mathcal{C}_1, ..., \mathcal{C}_m\}$. *The* logical implication problem *asks if* $\mathcal{T} \models \mathcal{C}$ *holds; that is, if all interpretations that satisfy each constraint in* $\mathcal{T}$ *must also satisfy* $\mathcal{C}$ *(the posed question).*

**Notation 4** We simplify the notation for path expressions in the rest of the paper by omitting the trailing $\mathsf{Id}$, and also allow a syntactic composition $\mathsf{Pf}_1 \,.\, \mathsf{Pf}_2$ of path expressions that stands for their concatenation.

**Definition 5 (Datalog$_{nS}^{\vee,\neg}$ [13])** *A* Datalog$_{nS}^{\vee,\neg}$ program *is a finite set of clauses constructed from monadic predicates* $P_i$ *and their negations, unary function symbols* $f_i$*, constants, and a single variable* $x$ *using the usual well-formedness rules. A* satisfiability problem *for a Datalog*$_{nS}^{\vee,\neg}$ *program* $\Pi$ *is to determine if* $\Pi$ *has a well-founded model.*

*A Datalog*$_{nS}$ *program is a finite set of* definite Horn *Datalog*$_{nS}^{\vee,\neg}$ *clauses. A* recognition problem *for a Datalog*$_{nS}$ *program* $\Pi$ *and a ground atom* $q$ *is to determine if* $q$ *is true in all models of* $\Pi$ *(i.e., if* $\Pi \cup \{\neg q\}$ *is not satisfiable).*

The unary function symbols are drawn from the same alphabet $F$ used for primitive attributes. Thus the path expressions in $\mathcal{DLFD}_{\text{reg}}$ and terms in Datalog$_{nS}^{\vee,\neg}$ correspond to each other.

**Notation 6** $\overline{\mathsf{Pf}}(t)$ denotes a Datalog$_{nS}^{\vee,\neg}$ term of the form "$f_k(\cdots f_1(t)\cdots)$". This notation corresponds to the $\mathsf{Pf} = f_1.\cdots.f_k$ notation for path functions.

It is known that a $\text{Datalog}^{\vee,\neg}_{nS}$ program has a model if and only if it has a Herbrand model [12,13,27]; this allows the use of syntactic techniques for model construction. To establish the complexity bounds we use the following result for the satisfiability of $\text{Datalog}^{\vee,\neg}_{nS}$ programs [13,18].

**Proposition 7** *The complexity of satisfiability for $\text{Datalog}^{\vee,\neg}_{nS}$ is DEXPTIME-complete. The lower bound holds for $\text{Datalog}_{nS}$ programs.*

For the remainder of the paper, we use general first-order syntax for $\text{Datalog}^{\vee,\neg}_{nS}$ formulas instead of the clausal syntax to improve readability. However, each of the $\text{Datalog}^{\vee,\neg}_{nS}$ programs in the paper can be presented in a clausal form.

## 3 Decision Procedures and Complexity Bounds

Relationships between $\mathcal{DLF}_{\text{reg}}$, $\mathcal{DLFD}_{\text{reg}}$ and $\text{Datalog}^{\vee,\neg}_{nS}$ are the concerns of this section. The main underlying idea lies in relating $\mathcal{DLFD}_{\text{reg}}$ concept descriptions to monadic predicates and in simulating $\mathcal{DLFD}_{\text{reg}}$ *structural properties* as assertions in $\text{Datalog}^{\vee,\neg}_{nS}$.

### 3.1 $\mathcal{DLF}_{\text{reg}}$: Lower Bounds

We first show that every $\text{Datalog}_{nS}$ recognition problem can be simulated by a $\mathcal{DLF}_{\text{reg}}$ implication problem.

**Definition 8** *Let $\Pi$ be a $\text{Datalog}_{nS}$ program and $P(\overline{\mathsf{Pf}}(0))$ a ground atom (a goal). We construct an implication problem for $\mathcal{DLF}_{\text{reg}}$ as follows: a terminology from clauses in $\Pi$,*

$$\mathcal{T}_\Pi = \{\, \forall\, \mathsf{Pf}_1\,.Q_1 \sqcap \ldots \sqcap \forall\, \mathsf{Pf}_k\,.Q_k \sqsubseteq \forall\, \mathsf{Pf}\,.P : \\ P(\overline{\mathsf{Pf}}(x)) \leftarrow Q_1(\overline{\mathsf{Pf}}_1(x)), \ldots, Q_k(\overline{\mathsf{Pf}}_k(x)) \in \Pi \,\},$$

*and the posed question from ground facts in $Q_i(\overline{\mathsf{Pf}}_i(0)) \in \Pi$ and the goal $G = P(\overline{\mathsf{Pf}}(0))$,*

$$\mathcal{C}_{\Pi,G} = \forall\, \mathsf{Pf}_1\,.Q_1 \sqcap \ldots \sqcap \forall\, \mathsf{Pf}_k\,.Q_k \sqsubseteq \forall\, \mathsf{Pf}\,.P,$$

*where, assuming $\overline{\mathsf{Pf}}(t) = f_k(\cdots f_1(t)\cdots)$, $\forall\, \mathsf{Pf}\,.D$ denotes $\forall f_1 \ldots \forall f_k.D$.*

**Theorem 9** *Let $\Pi$ be a $\text{Datalog}_{nS}$ program and $G$ a goal. Then $\Pi \models G \iff \mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.*

The lower complexity bound then follows from [13,18].

**Corollary 10** *The implication problem for $\mathcal{DLF}_{\text{reg}}$ is DEXPTIME hard; this remains true for the fragment of $\mathcal{DLF}_{\text{reg}}$ in which all concept constructors are conjunctions and restrictions of the form "$\forall f.D$".*

This complements the hardness result for path functional dependencies [24]: while that result was obtained in a class-free setting, here we need multiple class labels. Note that the complexity is inherently connected with a $\forall f.D$ construct. Without it, the problem reduces to propositional satisfiability which is NP-complete.

### 3.2 A Decision Procedure for $\mathcal{DLF}_{\mathbf{reg}}$

We now consider the other direction: can $\mathcal{DLF}_{\mathrm{reg}}$ in turn be (naturally) simulated by $\mathrm{Datalog}_{nS}^{\vee,\neg}$? In this subsection, we show that this is indeed possible. In particular, we show how to construct a $\mathrm{Datalog}_{nS}^{\vee,\neg}$ satisfiability problem that is equivalent to a given $\mathcal{DLF}_{\mathrm{reg}}$ implication problem. In the simulation, $\mathcal{DLF}_{\mathrm{reg}}$'s concept descriptions $D$ are modeled by monadic predicates $P_D(x)$. The construction proceeds in three steps. The first encodes the structural properties of regular expression sets as $\mathrm{Datalog}_{nS}^{\vee,\neg}$ assertions.

**Definition 11 (Constraints for Regular Path Expressions)** *Let $\langle N, F, R, S\rangle$ denote a right-linear grammar $G$ for a given regular language $L$ consisting of nonterminal symbols $N$ unique to $L$, productions $R$ and start symbol $S \in N$.*[1] *For formula $\varphi$ with one free variable $x$ not containing any predicate symbols of the form $N^{B,\psi}$ and $G$ we define a $\mathrm{Datalog}_{nS}^{\vee,\neg}$ program*

$$\Pi_{\mathcal{L}} = \left\{\forall x.N^{A,\varphi}(x) \leftarrow \mathcal{M}(\varphi,\alpha) : A \rightarrow \alpha \in G\right\}$$

*where*

$$\mathcal{M}(P,\alpha) = \begin{cases} \varphi(x) & \text{if } \alpha = \mathsf{Id}; \\ \varphi(f(x)) & \text{if } \alpha = f; \\ N^{B,\varphi}(f(x)) & \text{if } \alpha = fB. \end{cases}$$

Intuitively, the atomic formula $N^{A,\varphi}(t)$ is true for a term $t$ if and only if there is $\mathsf{Pf} \in \mathcal{L}(A)$ such that $\varphi(\overline{\mathsf{Pf}}(t))$ is true ($\mathcal{L}(A)$ is the language generated by $G$ from the nonterminal $A$). Note that it is essential to use the *minimal model semantics* to define $N^{A,\varphi}(x)$. However, the rules that define the atoms associated with $G$'s nonterminal symbols are $\mathrm{Datalog}_{nS}$ rules and thus a unique minimal model exists and can be equivalently defined by an explicit *least fixpoint formula*[2].

The second step encodes the structural properties of $\mathcal{DLF}_{\mathrm{reg}}$ as $\mathrm{Datalog}_{nS}^{\vee,\neg}$ assertions.

**Definition 12 ($\mathcal{DLF}_{\mathrm{reg}}$ Concept Formation Constraints)** *Let $D$, $D_1$, and $D_2$ range over concept descriptions and $f$ over attribute names. We define*

$$\Pi_{\mathcal{DLF}_{\mathrm{reg}}} = \Pi_{\mathcal{L}} \cup \left\{ \begin{array}{l} \forall x.(P_D(x) \vee P_{\neg D}(x)), \forall x.\neg(P_D(x) \wedge P_{\neg D}(x)) \\ \forall x.P_{D_1 \sqcap D_2}(x) \leftrightarrow (P_{D_1}(x) \wedge P_{D_2}(x)) \\ \forall x.P_{\exists L.D}(x) \leftrightarrow N^{S,P_D}(x)) \\ \forall x.P_{\forall L.D}(x) \leftrightarrow \neg N^{S,\neg P_D}(x)) \end{array} \right\}.$$

*where $S$ is the start symbol in the grammar for $L$.*

[1] A grammar is right regular iff each production is of the form $A \rightarrow \mathsf{Id}$, $A \rightarrow a$ or $A \rightarrow bB$, where $A$ and $B$ are nonterminal symbols and $a$ and $b$ are terminal symbols. It is well known that such a grammar exists for any regular expression [19].

[2] This fact guarantees total models for the theories in this paper and avoids difficulties connected with unfounded recursion in general $\mathrm{Datalog}_{nS}^{\vee,\neg}$ programs. For a comprehensive survey of semantics for disjunctive logic programs see [15,26,28]. A thorough exploration of that subject for $\mathrm{Datalog}_{nS}^{\vee,\neg}$ is beyond the scope of this paper.

The set $\Pi_{\mathcal{DLF}_{\text{reg}}}$ captures the structural relationships between $\mathcal{DLF}_{\text{reg}}$ concepts. Although the set is infinite in general, the set of concepts and regular path expressions appearing in a particular implication problem, $\mathcal{T} \models \mathcal{C}$, is finite. Hence, one can restrict the set of assertions in $\Pi_{\mathcal{DLF}_{\text{reg}}}$ to a finite subset $\Pi^{\mathcal{T},\mathcal{C}}_{\mathcal{DLF}_{\text{reg}}}$ that contains only predicates that define concepts and regular grammars in $\mathcal{T} \cup \{\mathcal{C}\}$. In the following, we omit the superscripts whenever clear from the context.

The translation of the inclusion constraints is the final third step in the overall translation of a $\mathcal{DLF}_{\text{reg}}$ implication problem.

**Definition 13** *Let $\mathcal{T}$ and $\mathcal{C} \equiv D \sqsubseteq E$ be a $\mathcal{DLF}_{\text{reg}}$ terminology and an inclusion constraint, respectively. We define*

$$\Pi_{\mathcal{T}} = \{\forall x.P_D(x) \rightarrow P_E(x) : D \sqsubseteq E \in \mathcal{T}\}$$

*and*

$$\Pi_{\mathcal{C}} = \{P_D(0), P_{\neg E}(0)\}.$$

The two clauses $\Pi_{\mathcal{C}}$ represent the skolemized version of $\neg\forall x.P_D(x) \rightarrow P_E(x)$; 0 is the Skolem constant for $x$. As usual, a model "containing" $\Pi_{\mathcal{C}}$ is a counterexample for $\mathcal{C}$.

**Theorem 14** *Let $\mathcal{T}$ and $\mathcal{C}$ be a $\mathcal{DLF}_{\text{reg}}$ terminology and inclusion dependency, respectively. Then $\mathcal{T} \models \mathcal{C} \iff \Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*[3]

This result shows that $\mathcal{DLF}_{\text{reg}}$ is essentially an alternative *variable-free* syntax for (monadic) Datalog$^{\vee,\neg}_{nS}$. Also, as a consequence we have:

**Corollary 15** *The implication problem for $\mathcal{DLF}_{\text{reg}}$ is DEXPTIME-complete.*

### 3.3 Adding Regular Functional Dependencies: $\mathcal{DLFD}_{\text{reg}}$

We now consider $\mathcal{DLFD}_{\text{reg}}$ which adds equality generating concepts to $\mathcal{DLF}_{\text{reg}}$. In this more general setting, an inclusion dependency that contains such a concept is called a *regular path functional dependency* (regular PFD). Such dependencies have the form $(D_1 \sqsubseteq D_2 : L_1 \rightarrow L_2)$.

There are two cases to consider that depend on the structure of $\mathcal{C}$ for an arbitrary $\mathcal{DLFD}_{\text{reg}}$ implication problem $\mathcal{T} \models \mathcal{C}$.

*Case 1:* $\mathcal{C}$ is not a regular PFD. In this case, it is straightforward to show that any regular PFDs occurring in $\mathcal{T}$ will not interfere with the decision procedure from Section 3.2.

**Lemma 16** *Let $\mathcal{T}'$ be the set of inclusion dependencies in a $\mathcal{DLFD}_{\text{reg}}$ terminology $\mathcal{T}$ that are not regular PFDs. Then if $\mathcal{T}'$ has a model it also has a Herbrand model. A Herbrand model is also a model of $\mathcal{T}$.*

[3] Note that we are assuming that satisfiability for Datalog$^{\vee,\neg}_{nS}$ is defined with respect to well-founded models.

Thus, the implication problem reduces to the problem in Section 3.2 since, by the above Lemma, the regular PFDs in $\mathcal{T}$ do not interfere with the decision procedure.

*Case 2:* $\mathcal{C}(= D_1 \sqsubseteq D_2 : L_1 \rightarrow L_2)$ is a regular PFD. To falsify $\mathcal{C}$, it must be possible to have *two* objects, one in $D_1$ and another in $D_2$, that satisfy the preconditions of the dependency but that fail to satisfy the conclusion. We therefore construct two copies of the interpretation for the pure $\mathcal{DLF}_{\text{reg}}$ constraints in $\mathcal{T}$ similarly to [21,33]. However, as Herbrand terms are essentially the same in the two copies, it is sufficient to distinguish them by renaming the predicate symbols [24]. In addition, we need to model the "rules" of equality and their interaction with concept descriptions. The structural rules for $\mathcal{DLFD}_{\text{reg}}$ are thus defined as follows:

$$\Pi_{\mathcal{DLFD}_{\text{reg}}} = \Pi^G_{\mathcal{DLF}_{\text{reg}}} \cup \Pi^B_{\mathcal{DLF}_{\text{reg}}} \cup \left\{ \begin{array}{l} \forall x.\mathsf{Eq}(x) \rightarrow \mathsf{Eq}(f(x)) \\ \forall x.\mathsf{Eq}(x) \rightarrow (P^G_D(x) \leftrightarrow P^B_D(x)) \end{array} \right\},$$

where $\Pi^G$ and $\Pi^B$ are sets of assertions $\Pi$ in which every predicate symbol $\rho$ is renamed to a "green" version $\rho^G$ and a "blue" version $\rho^B$, respectively.

**Definition 17** *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFD}_{\text{reg}}$ implication problem for which $\mathcal{C}$ is the regular PFD*

$$D_1 \sqsubseteq D_2 : L_1 \rightarrow L_2,$$

*let $\mathcal{T}''$ be all regular PFDs in $\mathcal{T}$, and let $\mathcal{T}' = \mathcal{T} \setminus \mathcal{T}''$. We define*

$$\Pi_{\mathcal{C}} = \{P^G_{D_1}(0), P^B_{D_2}(0), \neg N^{S_1,\neg\mathsf{Eq}}(0), N^{S_2,\neg\mathsf{Eq}}(0)\}$$

*and*

$$\Pi_{\mathcal{T}} = \Pi^G_{\mathcal{T}'} \cup \Pi^B_{\mathcal{T}'} \cup \left\{ \begin{array}{r} \forall x.(P^G_{D_i}(x) \wedge P^B_{D_j}(x) \wedge (\neg N^{S_i,\neg\mathsf{Eq}}(x))) \rightarrow \neg N^{S_j,\neg\mathsf{Eq}}(x) \\ \forall x.(P^B_{D_i}(x) \wedge P^G_{D_j}(x) \wedge (\neg N^{S_i,\neg\mathsf{Eq}}(x))) \rightarrow \neg N^{S_j,\neg\mathsf{Eq}}(x) \\ \text{for } (D_i \sqsubseteq D_j : L_i \rightarrow L_j) \in \mathcal{T}'' \end{array} \right\}$$

*where $S_i$ and $S_j$ are the start symbols in the grammars for $L_i$ and $L_j$, respectively.*

**Theorem 18** *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFD}_{\text{reg}}$ implication problem in which $\mathcal{C}$ is a regular PFD. Then $\mathcal{T} \models \mathcal{C}$ if and only if $\Pi_{\mathcal{DLFD}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*

And since all $\mathcal{DLF}_{\text{reg}}$ implication problems are also $\mathcal{DLFD}_{\text{reg}}$ implication problems, we have:

**Corollary 19** *The implication problem for $\mathcal{DLFD}_{\text{reg}}$ is DEXPTIME-complete.*

## 4 Structural Equality in XML

We now show how to map XML documents to $\mathcal{DLFD}_{\text{reg}}$ interpretations in a way that enables useful reasoning about the structural equality of arbitrary

subdocuments. To begin, it will be useful to have a concise definition of an XML document. Our formulation is based on the common practice of interpreting an XML document or a document collection as an ordered forest of rooted node-labeled ordered trees.

**Definition 20 (XML Forests and Trees)** *Let* String *be a set of strings. We define the set* XF *of XML forests inductively by*

$$\mathsf{XF} \quad = \quad [\,] \;\mid\; [\mathsf{XNode}(s, x)] \;\mid\; x@y,$$

*where $s$ is in* String*, $x$ and $y$ are in* XF*, $[\,]$ denotes an empty forest*[4]*, $[\mathsf{XNode}(s, x)]$ denotes a forest containing a single tree with a root labeled $s$ (the string $s$ represents a tag or* PCDATA*) and an ordered forest $x$, and $x@y$ denotes a concatenation of two ordered forests.*

This formulation of XML is very simple with no explicit accounting of node identity, of *element*, *attribute* or *text* node types, or of "don't care" child node order. However, such features can easily be added by additional encoding conventions that relate either to node labeling or to subtree patterns. A text leaf node with CDATA "abc", for example, might be encoded using the label "text:abc". A similar approach can be taken to represent attributes, etc.

Reasoning about XML documents in $\mathcal{DLFD}_{\mathrm{reg}}$ is achieved by mapping ordered forests corresponding to XML documents to $\mathcal{DLFD}_{\mathrm{reg}}$ *interpretations*. As in [9], we encode arbitrary XML forests by *binary trees* in which the *first* edge connects parent nodes with their first child and the *next* edge with their right sibling [25]. However, to be able to reason about structural equality, we add a third *label* edge connecting a node with its string label. Infinite completions of such tree yields an $\mathcal{DLFD}_{\mathrm{reg}}$ interpretation. Formally:

**Definition 21** *Let $\mathcal{F} \in \mathsf{XF}$ be a XML forest. We define an $\mathcal{DLFD}_{\mathrm{reg}}$ interpretation that represents this forest in two steps.*

1. *Let* String $\subset \Delta$ *be the set of all strings. For every document tag* <a> *we define a primitive class $C_{\mathtt{a}}$ interpreted by $(C_{\mathtt{a}})^{\mathcal{I}} = \{\texttt{<a>}\} \subset$ String.*
2. *The tree structure of the XML document is then captured by defining the interpretation for an additional primitive concept $C_{\mathrm{XML}}$, satisfying $(C_{\mathrm{XML}})^{\mathcal{I}} \cap$ String $= \emptyset$, and the interpretation of the primitive features $f$, $n$, and $l$. The interpretation of the primitive concept $C_{\mathrm{XML}}$ is defined by simultaneous induction on the structure of $\mathcal{F}$ utilizing partial interpretations $(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}}$ and an auxiliary $r_{\mathcal{F}}$ constant (denoting the root of the encoded document $\mathcal{F}$).*
   - *$\mathcal{F}$ is an empty forest. Then $(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}} = \emptyset$ and $r_{\mathcal{F}}$ is an arbitrary element $n \in \Delta -$ String.*
   - *$\mathcal{F}$ is a tree $\mathsf{XNode}(s, \mathcal{F}')$. Then*

$$(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}} := (C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}'} \cup \{n\} \quad \text{for} \quad n \in \Delta - \mathsf{String} \cup (C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}'}.$$

   *In addition, we modify the interpretation of the primitive features asserting that $(l)^{\mathcal{I}}(n) = s \in$ String*, *and $(f)^{\mathcal{I}}(n) = r_{\mathcal{F}'}$ and set $r_{\mathcal{F}} = n$.*

[4] We employ common list notation to represent ordered forests.

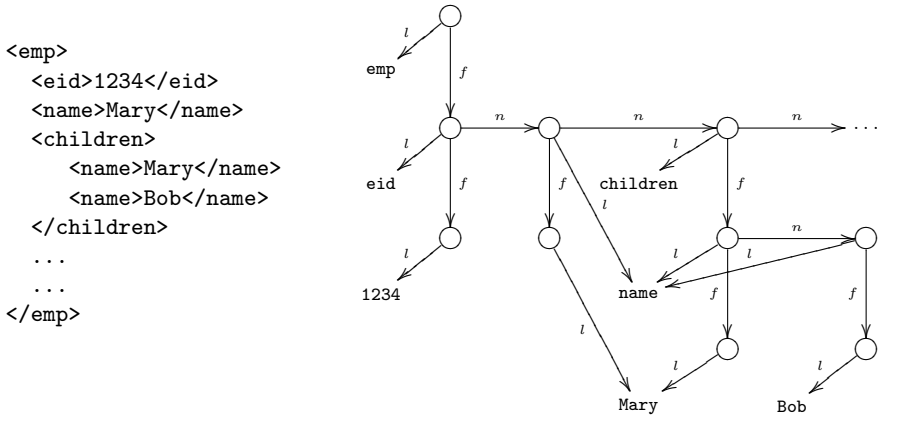



**Fig. 2.** An XML Document and $\mathcal{DLFD}_{\text{reg}}$ Interpretation

– *$\mathcal{F}$ is a forest of trees $\mathcal{T}_1\mathcal{T}_2\ldots\mathcal{T}_k$ such that $(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_i} \cap (C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_j} = \emptyset$.*[5] *Then*

$$(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{F}} := \bigcup_{0<i\leq k} (C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_i}.$$

*In addition, we modify the interpretation of the primitive features asserting that $(n)^{\mathcal{I}}(r_{t_i}) = r_{t_{i+1}}$ for $0 < i < k$ and set $r_{\mathcal{F}} = r_{t_1}$.*

Without loss of generality, we assume that feature values not explicitly defined in this construction are roots of complete ternary trees, the nodes of which do not belong to interpretations of primitive concepts defined above.

**Theorem 22** *Let $o_1, o_2 \in (C_{\text{XML}})^{\mathcal{I}}$ be two nodes in the interpretation $(\Delta, (.)^{\mathcal{I}})$ that correspond to the roots of XML forests $\mathcal{F}_1, \mathcal{F}_2$, respectively. Then $\mathcal{F}_1 = \mathcal{F}_2$ structurally if and only if $(\mathsf{Pf})^{\mathcal{I}}(o_1) = (\mathsf{Pf})^{\mathcal{I}}(o_2)$ for all $\mathsf{Pf} \in \mathcal{L}((f, n) * .l)$.*

**Example 23** In Figure 2, we illustrate an XML fragment together with the $\mathcal{DLFD}_{\text{reg}}$ interpretation that corresponds to this fragment. Now consider how to express that `<eid>`s are always integer values and are the first components of `<emp>`s. This is indirectly accomplished by including the following inclusion dependency in a terminology constraining $C_{\texttt{emp}}$:

$$\forall l.C_{\texttt{emp}} \sqsubseteq (\forall f.l.C_{\texttt{eid}}) \sqcap (\forall f.f.l.C_{\texttt{int}})$$

Finally, consider how to express that `<eid>`s are keys for employee subdocuments. This is also indirectly accomplished by including the following regular path functional dependency in the same terminology:

$$\forall l.C_{\texttt{emp}} \sqsubseteq (\forall l.C_{\texttt{emp}}) : f.f.l \rightarrow (f, n) * .l$$

[5] It is always possible to pick disjoint sets to interpret nodes in distinct trees.

Observe that this dependency requires no knowledge of the structure of employee subdocuments beyond the fact that an `<eid>` element is the first child of every such document.

Now, as a consequence of Theorem 22, we can replace the structural equality in the `where` clause of the XQuery in Example 1 by a much more efficient comparison of integer values:

```
where x=y      ⇒      where x/eid/data()=y/eid/data()
```

In addition the typing constraint above specifies *the location* of the `<eid>` component in the XML tree (the first child of `<emp>`) and the location (first child of `<eid>`) and structure (integer) of the identifier itself.

The *positional* nature of specifying structural relationships between elements is essential to reasoning about structural equality. In particular *keyword-based* functional dependencies (i.e., those based on element tag names) [2,3] cannot distinguish documents `<a>1</a><b>2</b>` and `<b>2</b><a>1</a>`.

## 5 Summary

Structural equality is an important performance issue for XML data models and query languages. We have presented a description logic called $\mathcal{DLFD}_{\text{reg}}$ that can be used for reasoning about structural equality in such models, and have outlined how $\mathcal{DLFD}_{\text{reg}}$ can be applied in the case of XML and XQuery. This application depends on a more powerful version of an $\mathsf{fd}$ concept constructor in $\mathcal{DLFD}_{\text{reg}}$ that has a novel and essential ability to express functional dependencies over sets of possibly infinite feature paths defined by regular languages. Thus, our work compliments earlier work [9] in which a description logic is used to reason about XML document type definitions.

We have also presented a decision procedure for $\mathcal{DLFD}_{\text{reg}}$. The procedure is based on mapping implication problems in $\mathcal{DLFD}_{\text{reg}}$ to satisfaction problems in the logic programming language $\text{Datalog}_{nS}^{\vee,\neg}$. It is worth noting that this can in turn be reduced to the classical decision problem for the Ackermann ($\exists^*\forall\exists^*$) prefix [1].

**Definition 24 (Monadic Ackerman Formulae)** *Let $P_i$ be monadic predicate symbols and $x, y_i, z_i$ variables. A* monadic first-order formula in the Ackermann class *is a formula of the form $\exists z_1 \ldots \exists z_k \forall x \exists y_1 \ldots \exists y_l . \varphi$ where $\varphi$ is a quantifier-free formula over the symbols $P_i$.*

To establish this final relationship, we can appeal to the results in [22] in which the authors outline a mapping of satisfiability problems for the $\mu$-calculus to a closed disjunctive fixpoint free fragment of the $\mu$-calculus. Note that it is straightforward to map $\text{Datalog}_{nS}^{\vee,\neg}$ satisfiability problems, in turn, to $\mu$-calculus when recursion in $\text{Datalog}_{nS}^{\vee,\neg}$ passes through even numbers of negations.

### 5.1 Future Work

There are several avenues of work currently under way that we believe will enhance the results of this paper. In particular, we are exploring the possibility of adapting results in [21] to allow the regular path functional dependencies in $\mathcal{DLFD}_{\text{reg}}$ to have empty left-hand-sides, a serious possibility in view of the fact that element tags in XML are not (a least apriori) "isa" related. Such constraints can be used to (almost) simulate the incorporation of nominals in a terminology. Another topic we are exploring relates to finite models. Although [21] has shown that any object model with path functional dependencies does not have the finite model property, we believe an acyclicity property that underlies XML document type definitions can be exploited to recover the finite model property for related terminologies.

So how "close to the cliff or the valley" have we come? One of the remaining limitations of $\mathcal{DLFD}_{\text{reg}}$ is the lack of an ability to define roles that are inverse attributes. This would represent a first opportunity for roles and functional dependencies to interact in $\mathcal{DLFD}_{\text{reg}}$. Although the details are beyond the scope of this paper, it is possible to adapt the undecidability result of [10] to show that $\mathcal{DLFD}_{\text{reg}}$ extended with inverse attributes will render its implication problem undecidable. However, some limited capacity to express inverse roles while ensuring decidability is still very desirable. Syntactic restrictions on regular path functional dependencies along the lines considered in [10] merit particular consideration. Second, the consequences of granting full first-order status to such dependencies are not clear. Indeed, the epistemological significance of either a negated fd or even the disjunction of two fds is unclear. Finally, we plan to investigate more general decidable constraint theories as the basis for path dependencies [4] and to integrate results in [32] that relate to ordering dependencies.

**Acknowledgments.** The authors gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada, the Communications and Information Technology of Ontario and Nortel Networks Ltd. for their support of this research.

## References

1. Wilhelm Ackermann. *Solvable Cases of the Decision Problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1954.
2. Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On Verifying Consistency of XML Specifications. In *ACM Symposium on Principles of Database Systems*, pages 259–270, 2002.
3. Marcelo Arenas and Leonid Libkin. Normal Form for XML Documents. In *ACM Symposium on Principles of Database Systems*, pages 85–96, 2002.
4. Marainne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-Generating Dependencies. In *International Conference on Database Theory*, 1995.

5. S. Boag, D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery 1.0: An XML Query Language. Technical report, W3C, 2001.
6. Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, 1997.
7. Alexander Borgida and Grant E. Weddell. Adding uniqueness constraints to description logics. In *International Conference on Deductive and Object Oriented Databases, DOOD*, pages 85–102, 1997.
8. Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. Keys for XML. In *World Wide Web*, pages 201–210, 2001.
9. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *Journal of Logic and Computation*, 9(1):295–318, 1999.
10. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *International Joint Conference on Artificial Intelligence*, pages 155–160, 2001.
11. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In *Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.
12. Jan Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols*. PhD thesis, Rutgers University, 1990. Laboratory for Computer Science Research LCSR-TR-142.
13. Jan Chomicki and Tomasz Imieliński. Finite Representation of Infinite Query Answers. *ACM Transactions on Database Systems*, 18(2):181–223, June 1993.
14. Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-solving using the `dlv` System. In Jack Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.
15. Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
16. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In *Principles of Knowledge Representation and Reasoning, KR'98*, pages 406–417, 1998.
17. Wenfei Fan and Leonid Libkin. On XML integrity constraints in the presence of DTDs. In *Symposium on Principles of Database Systems*, 2001.
18. Martin Fürer. Alternation and the Ackermann Case of the Decision Problem. *L'Enseignement Math.*, 27:137–162, 1981.
19. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
20. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical Reasoning for Expressive Description Logics. In *Logic Programming and Automated Reasoning, LPAR'99*, pages 161–180, 1999.
21. Minoru Ito and Grant Weddell. Implication Problems for Functional Constraints on Databases Supporting Complex Objects. *Journal of Computer and System Sciences*, 49(3):726–768, 1994.
22. David Janin and Igor Walukiewicz. Automata for the $\mu$-calculus and related results. In *Mathematical Foundations of Computer Science*, pages 552–562, 1995.
23. Vitaliy L. Khizder, David Toman, and Grant Weddell. Reasoning about Duplicate Elimination with Description Logic. In *Rules and and Objects in Databases, DOOD 2000 (part of Computational Logic 2000)*, pages 1017–1032, 2000.

24. Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *International Conference on Database Theory ICDT'01*, pages 54–67, 2001.
25. Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Welsley (2ed), 1998.
26. Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.
27. John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
28. Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA, 1992.
29. Alberto O. Mendelzon and Peter T. Wood. Functional Dependencies in Horn Clause Queries. *TODS*, 16(1):31–55, 1991.
30. Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.
31. David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental Techniques for Order Optimization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 57–67, 1996.
32. David Toman and Grant E. Weddell. On Attributes, Roles, and Dependencies in Description Logics and the Ackermann Case of the Decision Problem. In *Proceedings of Description Logics, CEUR-WS, vol.49*, pages 76–85, 2001.
33. Grant E. Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *TODS*, 17(1):32–64, 1992.

## A Adding Roles to $\mathcal{DLF}_{\mathbf{reg}}$

Traditionally, description logics allow *roles*—binary relations between concepts. However, while general Ackermann formulae allow arbitrary arity relations in their matrix, they still require the use of a *single* universal ($\forall$) quantifier in their prefix. This prevents a direct formulation of the $\forall R.D$ concept since two $\forall$'s are needed (then adding unary function symbols leads to undecidable theories [6]).

Therefore we use a less direct formulation by modeling roles in $\mathcal{DLFR}$ via attributes. The essential problem is that $\exists R.D_i$ concepts can force a single object to be related via a role $R$ to multiple objects satisfying different constraints $D_i$, that, in general, may be disjoint. However, as all concept descriptions are essentially monadic predicates, one can syntactically determine the maximal number of such objects needed for a given implication problem.

**Definition 25 ($\exists$-rank)** *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFR}$ implication problem. The number of distinct occurrences of $\exists R.D$ in $\mathcal{T} \cup \{\mathcal{C}\}$ is denoted* $\mathrm{Rank}(\mathcal{T}, \mathcal{C})$.

Now let $\mathcal{T}$ and $\mathcal{C}$ be fixed, and $\rho$ and $\delta_0, \ldots, \delta_l$, where $l = \mathrm{Rank}(\mathcal{T}, \mathcal{C})$, be function symbols neither in $\mathcal{T}$ nor in $\mathcal{C}$. We model a role $R$ by a monadic predicate $P_R$. The fact that $(o, o') \in (R)^{\mathcal{I}}$, for $o, o' \in \Delta$, is captured by asserting $P_R(\delta_i(o))$ and $o' = \rho(\delta_i(o))$ for some $0 \leq i \leq l$. The new predicates $P_R$ (and function symbols $\delta_i$ and $\rho$) are constrained to simulate the behavior of the $\forall R.D$ and $\exists R.D$ concepts by the following assertions:

$$\Pi_{\mathcal{DLFR}} = \Pi_{\mathcal{DLF}_{\text{reg}}} \cup \left\{ \begin{array}{l} \forall x.P_{\exists R.D}(x) \leftrightarrow \bigvee_{j=0}^{l} [P_R(\delta_j(x)) \wedge P_D(\rho(\delta_j(x)))] \\ \forall x.P_{\forall R.D}(x) \leftrightarrow \bigwedge_{j=0}^{l} [P_R(\delta_j(x)) \rightarrow P_D(\rho(\delta_j(x)))] \end{array} \right\}.$$

For any fixed implication problem of size $n$ the size of the assertions is $O(n^2)$.

**Theorem 26** *Let $\mathcal{T}$ and $\mathcal{C}$ be a $\mathcal{DLFR}$ terminology and inclusion dependency. Then $\mathcal{T} \models \mathcal{C} \iff \Pi_{\mathcal{DLFR}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable. The later can be decided in DEXPTIME.*

Note the similarity with the *tree* models for $\mathcal{DLR}$: essentially, one can convert a dag induced by roles into a tree by replacing $(o_1, o_2) \in (R)^{\mathcal{I}}$ by $(o_1, o_2')$ and $(o_1', o_2)$ where $o_1$ and $o_1'$ ($o_2$ and $o_2'$, respectively) belong to the same concepts. Such an interpretation is still a model.

### A.1 Role Inverses and Other Role Constructs

To model an inverse role $R^{-1}$, we need to modify the definitions for $P_{\forall R.D}$ and $P_{\exists R.D}$ to take account of the fact that if a parent of an object (i.e., the argument of the function simulating the role) is related to a $\forall R.D$ object via an inverse $R^{-1}$, then this object must be related to the parent via the original role $R$. Thus the parent must satisfy $D$ (the $\exists R.D$ argument is similar). These observations are captured by the assertions (similarly for $P_{\forall R^{-1}.D}$ and $P_{\exists R^{-1}.D}$):

$$\begin{array}{lll} \forall x.P_{\exists R.D}(x) & \leftrightarrow \bigvee_{j=0}^{l} [P_R(\delta_j(x)) \wedge P_D(\rho(\delta_j(x)))] & \text{for } x \neq \rho(\delta_i(y)) \\ \forall x.P_{\exists R.D}(\rho(\delta_i(x))) & \leftrightarrow \bigvee_{j=0}^{l} [P_R(\delta_j(\rho(\delta_i(x)))) \wedge P_D(\rho(\delta_j(\rho(\delta_i(x)))))] & \\ & \quad \vee [P_{R^{-1}}(\delta_i(x)) \wedge P_D(x)] & \text{otherwise} \\ \forall x.P_{\forall R.D}(x) & \leftrightarrow \bigwedge_{j=0}^{l} [P_R(\delta_j(x)) \rightarrow P_D(\rho(\delta_j(x)))] & \text{for } x \neq \rho(\delta_i(y)) \\ \forall x.P_{\forall R.D}(\rho(\delta_i(x))) & \leftrightarrow \bigwedge_{j=0}^{l} [P_R(\delta_j(\rho(\delta_i(x)))) \rightarrow P_D(\rho(\delta_j(\rho(\delta_i(x)))))] & \\ & \quad \wedge [P_{R^{-1}}(\delta_i(x)) \rightarrow P_D(x)] & \text{otherwise} \end{array}$$

Note that the $x \neq \rho(\delta_i(y))$ condition can be eliminated by enumerating all terms that do not have the above form (i.e., do not start with $\rho\delta_i$). Such an enumeration is finite (still $O(n^2)$). Theorem 26 and its proof naturally extend to this setting. Other constructs that involve roles, such as *numerical restrictions*, *role hierarchies*, *role constructors*, *roles with arity* $\geq 2$, and their combinations, can be similarly captured by appropriate assertions over the monadic predicates that model roles as long as the DL dialect itself has a tree model property [11, 20].

# Containment of Aggregate Queries*

Sara Cohen[1], Werner Nutt[2], and Yehoshua Sagiv[1]

[1] School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
{sarina,sagiv}@cs.huji.ac.il
[2] School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh EH14 4AS
Scotland, UK
nutt@macs.hw.ac.uk

**Abstract.** The problem of deciding containment of aggregate queries is investigated. Containment is reduced to equivalence for queries with *expandable* aggregation functions. Many common aggregation functions, such as *max*, *cntd* (count distinct), *count*, *sum*, *avg*, *median* and *stdev* (standard deviation) are shown to be expandable. It is shown that even in the presence of integrity constraints, containment can be reduced to equivalence. For conjunctive *count* and *sum*-queries, simpler characterizations for containment are given, that do not require checking equivalence. These results are built upon in order to solve the problem of finding maximally-contained sets of rewritings for conjunctive *count*-queries.

## 1 Introduction

Rewriting queries using views is a fundamental problem in databases. Research in view usability attempts to answer the questions "what can be computed from what, and how?" Such techniques have applications in a number of areas. In query optimization, the execution of a query can be accelerated if results from previous queries can be used to compute answers [23,2]. In designing information systems that should periodically process a huge number of a priori known queries, it can be beneficial to store beforehand intermediate results that can be useful for as many queries as possible [17,20]. In integrating heterogeneous information sources, one approach is to model the contents and the relationships of a set of sources by setting up one rich schema and to describe the sources as views on this schema. In order to answer queries issued against the schema we must rewrite them using the "views" [16].

While the focus of this work was for a long time on non-aggregate queries, interest in aggregate queries has been motivated recently by the rapid expansion of data warehousing and decision support, where aggregate queries typically occur. Optimization based on the reuse of previously computed results is particularly

* This work was supported by the Israel Science Foundation (Grant 96/01-1) and the British EPSRC (Grant GR/R74932/01).

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 111–125, 2003.

promising for aggregate queries, since often a huge amount of data is touched to produce a single aggregate value. In fact, most existing data warehouses make use of this idea in a rather ad hoc way in their optimization algorithms [13].

There are several papers that provide characterizations for equivalence of aggregate queries [18,6,9,4]. Algorithms for rewriting aggregate queries have been presented [6,7,10,21]. However, these algorithms investigated the problem of finding a rewriting that is *equivalent* to the given query. They also did not use unions of aggregate queries to rewrite aggregate queries and thus, could not find rewritings in some cases, even though such rewritings existed.

This paper investigates the problem of deciding containment of aggregate queries. To the best of our knowledge, there are no previous results on this topic. See Section 5 for related work on containment of non-aggregate queries. We show how it is possible to reduce containment of a wide class of aggregate queries to equivalence. We build on our containment results and present a method for finding maximally-contained sets of rewritings of *count*-queries. Maximally-contained sets of rewritings are of importance in the context of information integration. This solves an open problem mentioned in [14].

Section 2 contains basic definitions. We present our containment results in Section 3. A method for computing maximally-contained sets of rewritings is detailed in Section 4. Section 5 concludes. An extended version of this paper, containing proofs and additional examples, is available on the Web [5].

## 2 Aggregate Queries

We introduce aggregate queries and review their basic properties. Our aggregate queries can contain disjunction, negation and comparisons.

### 2.1 Syntax of Queries

Let $P$ be a fixed relational vocabulary and let $C$ be a set of constants. We assume that there is a linear ordering defined over $C$, which is either dense (like the rational numbers) or discrete (like the integers). We will usually define a *database* $\mathcal{D}$ as a *set* of ground atoms of the form $p(c_1, \ldots, c_k)$, where $p \in P$ is a predicate of arity $k$ and $c_1, \ldots, c_k \in C$. At times we will consider databases that are *bags* of ground atoms, but then we will state this explicitly.

A *condition* $\varphi(\bar{z}, \bar{x})$, where $\bar{z}$, $\bar{x}$ are tuples of variables, is a conjunction of positive and negated atomic formulas over $P$, $<$ and $\leq$, with constants from $C$. The comparisons are interpreted w.r.t. the ordering over $C$. The variables in $\bar{x}$ are exactly the *free* variables in $\varphi(\bar{z}, \bar{x})$ and the variables in $\bar{z}$ are *bound* by existential quantifiers. A *query* has the form

$$q(\bar{x}) \leftarrow \varphi_1(\bar{z}_1, \bar{x}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x})$$

where $\varphi_i(\bar{z}_i, \bar{x})$ is a condition, for all $i$. We call $q(\bar{x})$ the *head* of the query and $\varphi_1(\bar{z}_1, \bar{x}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x})$ the *body* of the query. Each condition $\varphi_i(\bar{z}_i, \bar{x})$ is a

*disjunct* of the query. We sometimes use the notation $q(\bar{x})$ or $q$ as a short-hand for denoting the query above. We assume that all queries are *safe* [22]. Note that all variables not appearing in $\bar{x}$ are implicitly existentially quantified.

We assume that there is a fixed set of constants $C_{\text{agg}}$. If $S$ is a domain, we denote by $\mathcal{M}(S)$ the set of finite bags over $S$. A *$k$-ary aggregation function* is a function $\alpha\colon \mathcal{M}(C^k) \to C_{\text{agg}}$ that maps bags of $k$-tuples of values in $C$ to values in $C_{\text{agg}}$. In this paper we consider a wide class of aggregation functions, with the property of being *expandable*. Such aggregation functions include the unary aggregation functions *max*, *sum*, *avg*, *median*, *stdev* and *cntd* which map a bag of real numbers to the maximum, sum, average, median, standard deviation or number of distinct elements.[1] We also consider the expandable aggregation function *count* which maps a bag of tuples to its cardinality.

An *aggregate query* has the form $q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$, where $\alpha$ is an aggregation function and $\bar{x}$ and $\bar{y}$ are free variables appearing in $\varphi_i$, for all $i$. We call $\bar{x}$ the *grouping variables* and $\bar{y}$ the *aggregation variables*. We will also refer to the query $q$ as an *$\alpha$-query* since the aggregation function $\alpha$ appears in $q$'s head.

We distinguish the class of conjunctive queries. A query $q$ is conjunctive if its body contains a single disjunct $\varphi(\bar{z}, \bar{x})$ that is a conjunction of *positive* atomic formulas over $P$ and $C$. Note that conjunctive queries do not contain comparisons.

*Example 2.1.* Suppose that our database contains the relations `study(student, course, grade)` and `teach(prof, course)`. The conjunctive query $q$ finds the average grade of the students in Prof. Lau's courses:

$$q(c, avg(g)) \leftarrow \texttt{study}(s, c, g) \wedge \texttt{teach}(\texttt{Lau}, c)\,.$$

### 2.2 Semantics of Queries

We define in which way an aggregate query $q$, for a database $\mathcal{D}$, gives rise to a *set* of tuples $q^{\mathcal{D}}$. Aggregate queries are evaluated in two phases. In the first phase, the query retrieves a bag of tuples from the database. The tuples are then grouped into equivalence classes, and an aggregation function is applied to each equivalence class.

An *assignment* $\gamma$ for a condition $\varphi(\bar{z}, \bar{x}, \bar{y})$ is a mapping of the variables in $\bar{z}, \bar{x}, \bar{y}$ to constants in $C$. *Satisfaction* of a condition by an assignment w.r.t. a database is defined in the usual way.

Consider an aggregate query $q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$. Let $\Gamma_i$ be the set of satisfying assignments of $\varphi_i(\bar{z}_i, \bar{x}, \bar{y})$ with respect to $\mathcal{D}$. For a tuple of values $\bar{d}$ from $C$, let $\Gamma_{i,\bar{d}}$ be the set of assignments in $\Gamma_i$ that map $\bar{x}$ to $\bar{d}$, i.e.,

$$\Gamma_{i,\bar{d}} := \{\gamma \in \Gamma_i \mid \gamma(\bar{x}) = \bar{d}\}.$$

[1] Results for *min* are analogous to results for *max* and, thus, are not presented.

In the sets $\Gamma_{i,\bar{d}}$, we collect those satisfying assignments of $\varphi_i$ that agree on $\bar{d}$.

For each set $\Gamma_{i,\bar{d}}$, we define $\Gamma_{i,\bar{d}}(\bar{y})$ to be the bag of tuples of constants for $\bar{y}$ derived by applying the assignments in $\Gamma_{i,\bar{d}}$ to $\bar{y}$. Formally,

$$\Gamma_{i,\bar{d}}(\bar{y}) := \{\!\{\gamma(\bar{y}) \mid \gamma \in \Gamma_{i,\bar{d}}\}\!\}$$

is the bag of tuples obtainable by restricting assignments in $\Gamma_{i,\bar{d}}$ to $\bar{y}$.

Now we define the result of evaluating $q$ over $\mathcal{D}$, denoted $q^{\mathcal{D}}$ as

$$\left\{(\bar{d}, \alpha(\Gamma_{1,\bar{d}}(\bar{y}) \uplus \cdots \uplus \Gamma_{n,\bar{d}}(\bar{y}))) \mid \bar{d} = \gamma(\bar{x}) \text{ for some } i \text{ and } \gamma \in \Gamma_i\right\} \qquad (1)$$

where $\uplus$ is the bag union operator. Note that we have computed the function $\alpha$ on the bag corresponding to each mapping of $\bar{x}$ to constants. Observe that if a mapping satisfies more than one disjunct, then it contributes more than one value to the bag to which we apply $\alpha$. Thus, for example, the queries $q(x, sum(y)) \leftarrow a(x,y) \vee a(x,y)$ and $q'(x, sum(y)) \leftarrow a(x,y)$ are not equivalent since $q$ will always return an aggregate value twice that of $q'$.

For the special case of conjunctive $\alpha$-queries, we also consider evaluating queries over databases that are *bags* of ground atoms. Queries evaluated over a database defined as a bag are said to be evaluated under *bag-semantics*. Otherwise, the evaluation is under *set-semantics*. When evaluating a conjunctive $\alpha$-query under bag-semantics, we proceed as above (i.e., as under set semantics), except that there is only one $\Gamma_{\bar{d}}$ for each $\bar{d}$ (since there is only one disjunct) and $\Gamma_{\bar{d}}$ can be a bag of assignments. Consider the conjunctive $\alpha$-query

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow a_1(\bar{w}_1) \wedge \cdots \wedge a_k(\bar{w}_k),$$

where $\bar{w}_i$ are variables from $\bar{x}$, $\bar{y}$ or are bound variables. Let $\gamma$ be an assignment that satisfies the body of $q$ and maps $\bar{x}$ to $\bar{d}$. Suppose that $a_i(\gamma(\bar{w}_i))$ appears in $\mathcal{D}$ a total of $n_i$ times. Then, $\gamma$ will be in $\Gamma_{\bar{d}}$ a total of $n_1 \times \cdots \times n_k$ times. The rest of the computation proceeds as in Equation (1). Note that the output of an aggregate query is always a set, even if it is evaluated over a bag database.

### 2.3 Containment and Equivalence

Containment and equivalence of aggregate queries are defined in the natural way. An aggregate query $q$ is *contained* in an aggregate query $q'$ if over every database the set of results returned by $q$ is a subset of the results returned by $q'$. Formally, $q$ is contained in $q'$, denoted $q \subseteq q'$, if $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$ for all databases $\mathcal{D}$. Two queries $q$ and $q'$ are *equivalent*, denoted $q \equiv q'$, if over every database they return the same sets of results. Obviously, two queries are equivalent if and only if they contain each other.

For conjunctive queries $q$ and $q'$, we say that $q$ is *bag-contained* in $q'$, denoted $q \subseteq_{\mathrm{b}} q'$, if over every bag database $\mathcal{D}$, we have $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$. We say that $q$ and $q'$ are *bag-equivalent*, denoted $q \equiv_{\mathrm{b}} q'$, if $q \subseteq_{\mathrm{b}} q'$ and $q' \subseteq_{\mathrm{b}} q$.

## 3 Containment of Aggregate Queries

This section contains characterizations of containment among queries with a wide class of aggregation functions. In Section 4 we show how our containment results can be used to create rewritings of queries using views. Our criteria involve reducing containment of aggregate queries to equivalence of aggregate queries.

### 3.1 Queries with Expandable Aggregation Functions

We present the class of *expandable aggregation functions*. We show that they are common and reduce containment of queries with expandable aggregation functions to equivalences of queries.

Let $B$ be a bag of constants and $c$ be a constant. We denote by $|B|_c$ the multiplicity of $c$ in $B$. Given a non-negative integer $n$, we define the *$n$-expansion* of $B$. Intuitively, the $n$-expansion of $B$, denoted $B \otimes n$, is the bag that contains the same constants as those in $B$, but each constant has a multiplicity $n$ times larger than in $B$. Formally, for all constants $c$, $|B \otimes n|_c = |B|_c \times n$.

Aggregation functions can be characterized by their behavior on expanded bags. We say that an aggregation function $\alpha$ is *expandable* if for all bags $B$ and $B'$ and for all positive integers $n$ we have

$$\alpha(B \otimes n) = \alpha(B' \otimes n) \quad \Longleftrightarrow \quad \alpha(B) = \alpha(B')\,.$$

**Proposition 3.1 (Expandable Functions).** *The aggregation functions max, cntd, count, sum, avg, stdev and median are expandable.*

However, not all aggregation functions are expandable.

*Example 3.2.* The function *prod* (product) is not expandable. As an example, consider the bags $B = \{\!\{-2\}\!\}$ and $B' = \{\!\{2, 1\}\!\}$. Clearly, $prod(B) \neq prod(B')$. However, $prod(B \otimes 2) = prod(B' \otimes 2) = 4$. Hence, *prod* is not expandable. Observe that *prod* is expandable when the domain of the bags contains only non-negative numbers. This example also shows that the function *parity* is not expandable.

Given the aggregate queries

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_1(\bar{z}_1, \bar{x}, \bar{y}) \vee \cdots \vee \varphi_n(\bar{z}_n, \bar{x}, \bar{y})$$
$$q'(\bar{x}, \alpha(\bar{y}')) \leftarrow \varphi'_1(\bar{z}'_1, \bar{x}, \bar{y}') \vee \cdots \vee \varphi'_m(\bar{z}'_m, \bar{x}, \bar{y}')$$

we say that a query $p$ is a *product* of $q$ and $q'$ if $p$ is defined as

$$p(\bar{x}, \alpha(\bar{y})) \leftarrow \bigvee_{1 \leq i \leq n,\ 1 \leq j \leq m} \varphi_i(\bar{z}_i, \bar{x}, \bar{y}) \wedge \varphi'_j(\theta(\bar{z}'_j), \bar{x}, \theta(\bar{y}'))\,,$$

where $\theta$ is a substitution that maps the variables in $\bar{y}'$ and $\bar{z}'_j$ to distinct unused variables. A product of $q$ and $q'$ is essentially a join on the grouping variables

of $q$ and $q'$. Note that two products of $q$ and $q'$ are only distinguished by the substitution $\theta$ and thus, are isomorphic. Slightly abusing notation, we will write $q \otimes q'$ to refer to an arbitrary product of $q$ and $q'$. This is justified because the specific variables occurring in a query are never important for our arguments.

Containment of aggregate queries can be reduced to equivalence of appropriate aggregate queries, if the aggregation function is expandable.

**Theorem 3.3 (Containment).** *Consider the aggregate queries $q(\bar{x}, \alpha(\bar{y}))$ and $q'(\bar{x}, \alpha(\bar{y}'))$. Suppose that $\alpha$ is an expandable function. Then for any database $\mathcal{D}$*

$$q^{\mathcal{D}} \subseteq q'^{\mathcal{D}} \iff (q \otimes q)^{\mathcal{D}} = (q' \otimes q)^{\mathcal{D}}.$$

*Proof. (Sketch)* Consider a database $\mathcal{D}$ and a tuple $\bar{d}$. Suppose that $q$ computes the bag $B$ of values for $\bar{d}$ and $q'$ computes the bag $B'$ of values for $\bar{d}$. It is not difficult to show that in this case, $q \otimes q$ will compute the bag $B \otimes |B|$ for $\bar{d}$ and $q' \otimes q$ will compute the bag $B' \otimes |B|$ for $\bar{d}$. Since $\alpha$ is an expandable aggregation function, $\alpha(B) = \alpha(B')$ if and only if $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, for $|B| > 0$.

"$\Leftarrow$" Suppose that $q \otimes q \equiv q' \otimes q$. If $q$ returns an aggregate value for $\bar{d}$, then $|B| > 0$. Therefore, $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$ implies that $\alpha(B) = \alpha(B')$, i.e., $q$ and $q'$ return the same aggregate value for $\bar{d}$.

"$\Rightarrow$" Suppose that $q \subseteq q'$. If $q$ does not return an aggregate value for $\bar{d}$, then both $q \otimes q$ and $q' \otimes q$ will not return an aggregate value. Otherwise, $q$ returns an aggregate value for $\bar{d}$, and $q'$ returns the same aggregate value. Therefore, from $\alpha(B) = \alpha(B')$, we conclude that $\alpha(B \otimes |B|) = \alpha(B' \otimes |B|)$, i.e., $q \otimes q$ and $q' \otimes q$ return the same value. □

**Corollary 3.4 (Decidability).** *Checking if $q$ is contained in $q'$ is decidable if*

- *$q$ and $q'$ are count, sum or max-queries* or
- *$q$ and $q'$ are conjunctive avg or cntd-queries without constants.*

*Proof.* This follows from Theorem 3.3 and the decidability of equivalence for the classes above (see [18,6,9,4]). □

**Theorem 3.5 (Bag-Containment).** *Let $\alpha$ be an expandable function and $q$ and $q'$ be conjunctive $\alpha$ queries. Then, $q \subseteq_{\mathrm{b}} q'$ if and only if $q \otimes q \equiv_{\mathrm{b}} q' \otimes q$.*

*Proof.* The proof is analogous to the proof of Theorem 3.3. □

If $\alpha$ is not an expandable function, the reduction may not be correct.

*Example 3.6.* Consider the queries

$$q(\mathit{parity}(y)) \leftarrow p(y) \vee p(y)$$
$$q'(\mathit{parity}(y)) \leftarrow p(y)\,.$$

Let $k$ be the number of values $d$ such that $\mathcal{D}$ contains $p(d)$. Then, $q$ always returns the value "even", since it evaluates the parity of a bag with $2k$ values.

The query $q'$ returns the parity of $k$ since it evaluates the parity of a bag with $k$ values. Thus, $q \not\subseteq q'$, since if $k$ is odd, the queries $q$ and $q'$ will return different values. Consider the queries $q \otimes q$ and $q' \otimes q$. Clearly, $q \otimes q$ will always return the value "even", since the aggregation function *parity* is evaluated over a bag with cardinality $4k^2$. Similarly, $q' \otimes q$ will always return the value "even", since it computes the parity of a bag of size $2k^2$. Therefore, the queries $q \otimes q$ and $q' \otimes q$ are equivalent, even though $q \not\subseteq q'$.

### 3.2 Integrity Constraints

Even in the presence of arbitrary integrity constraints it is possible to reduce containment of aggregate queries to equivalence. We write $q \subseteq_I q'$ if for all databases $\mathcal{D}$ satisfying the integrity constraints $I$, it holds that $q^{\mathcal{D}} \subseteq {q'}^{\mathcal{D}}$. Similarly, we write $q \equiv_I q'$ if $q \subseteq_I q'$ and $q' \subseteq_I q$.

**Corollary 3.7 (Reduction w.r.t. Integrity Constraints).** *Consider the $\alpha$-queries $q$ and $q'$ and the integrity constraints $I$. Suppose that $\alpha$ is an expandable aggregation function. Then $q \subseteq_I q'$ if and only if $(q \otimes q) \equiv_I (q' \otimes q)$.*

*Proof.* Theorem 3.3 allows us to reduce containment with respect to a set of integrity constraints $I$ to equivalence with respect to $I$. □

It is possible to extend previous characterizations of equivalence of queries from [18,6,4] to equivalence with respect to a set of functional dependencies $F$. For example, it can be shown that if $q$ and $q'$ are conjunctive *count*-queries, then $q \equiv_F q'$ if and only if the chases of $q$ and $q'$ with respect to $F$ are isomorphic.

**Corollary 3.8 (Decidability).** *Given a set of functional dependencies $F$, the problem of checking whether $q \subseteq_F q'$ is decidable if $q$ and $q'$ are count, sum or max-queries.*

*Example 3.9.* Consider the queries $q$ and $q'$ which find the number of visitor advisors of a student and the number of advisors of a student, respectively:

$$q(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{visitor}(p)$$
$$q'(s, count) \leftarrow \texttt{advise}(p, s)\,.$$

In general, $q \not\subseteq q'$. Suppose that every student can have at most one advisor, i.e., the first column of `advise` is functionally dependent on the second. Let $F$ be the set of functional dependencies that contains exactly this dependency. We will show that that $q \subseteq_F q'$ even though applying the chase to $q$ and $q'$ has no effect. To prove that $q \subseteq_F q'$, we consider the products $q \otimes q$ and $q' \otimes q$, i.e.,

$$(q \otimes q)(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{visitor}(p) \wedge \texttt{advise}(p', s) \wedge \texttt{visitor}(p')$$
$$(q' \otimes q)(s, count) \leftarrow \texttt{advise}(p, s) \wedge \texttt{advise}(p', s) \wedge \texttt{visitor}(p').$$

By Corollary 3.7, $q \subseteq_F q'$ if and only if $(q \otimes q) \equiv_F (q' \otimes q)$. Indeed, applying the chase to $(q \otimes q)$ and $(q' \otimes q)$ results in isomorphic, and thus, equivalent queries.

### 3.3 Containment of Conjunctive Count-Queries

We show how the results in the previous section give rise to simpler characterizations for conjunctive *count*-queries.

A *bound atom* is an atom with at least one argument that is a bound variable. Otherwise, the atom is *free*. We sometimes write a conjunctive query as

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \varphi_f(\bar{x}, \bar{y}) \wedge \varphi_b(\bar{z}, \bar{x}, \bar{y})\,.$$

where $\varphi_f$ is a conjunction of free atoms and $\varphi_b$ is a conjunction of bound atoms. Note that for the special case of *count*-queries, $\bar{y}$ is an empty tuple of variables.

**Theorem 3.10 (Conjunctive Count-Queries).** *For the conjunctive queries*

$$q(\bar{x}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$$
$$q'(\bar{x}, count) \leftarrow \varphi'_f(\bar{x}) \wedge \varphi'_b(\bar{z}', \bar{x}).$$

*it holds that* $q \subseteq_{\mathrm{b}} q'$ *if and only if* $q \equiv_{\mathrm{b}} q'$*, and* $q \subseteq q'$ *if and only if both the following conditions hold*

- *the set of atoms in* $\varphi'_f(\bar{x})$ *is contained in the set of atoms in* $\varphi_f(\bar{x})$ and
- $\varphi'_b(\bar{z}', \bar{x})$ *is isomorphic to* $\varphi_b(\bar{z}, \bar{x})$.[2]

When considering the general problem of aggregate query containment, we assumed that both queries had the same grouping variables. This was without loss of generality since equalities between grouping variables and between grouping variables and constants could be expressed using comparisons in the bodies of the queries. In this section we discuss only conjunctive *count*-queries and such queries do not have comparisons in their bodies. Therefore, we characterize containment of queries with different grouping terms. (Both queries have the same number of grouping terms, however.) A query has the form

$$q(\bar{s}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})\,,$$

where $\bar{s}$ is a tuple of terms and $\bar{x}$ are exactly all the variables in $\bar{s}$.

Let $q(\bar{s}, count) \leftarrow \varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$ and $q'(\bar{s}', count) \leftarrow \varphi'_f(\bar{x}') \wedge \varphi'_b(\bar{z}', \bar{x}')$ be queries. A mapping $\theta$ from the variables in $q'$ to the terms in $q$ is a *containment mapping* if the following conditions hold

1. $\theta(\bar{s}') = \bar{s}$, i.e., $\theta$ maps the head of $q'$ to the head of $q$;
2. $\theta$ is injective on the bound variables in $q'$;
3. the set of atoms in $\theta(\varphi'_b(\bar{z}', \bar{x}'))$ is the same as the set of atoms in $\varphi_b(\bar{z}, \bar{x})$, i.e., $\theta$ maps bound atoms in $q'$ to bound atoms in $q$;
4. $\theta(\varphi'_f(\bar{x}'))$ is contained in $\varphi_f(\bar{x}) \wedge \varphi_b(\bar{z}, \bar{x})$, i.e., the images of the free atoms in $q'$ appear in $q$.

The following corollary follows from Theorem 3.10 and will be used in Section 4 when discussing rewritings of queries.

**Corollary 3.11 (Conjunctive Count-Queries (2)).** *Let* $q$ *and* $q'$ *be conjunctive count-queries, possibly with different terms in their heads. Then* $q \subseteq q'$ *if and only if there is a containment mapping from* $q'$ *to* $q$.

[2] Note that our isomorphisms ignore duplicate occurrences of atoms in a conjunction.

### 3.4 Containment of Conjunctive Sum-Queries

Theorem 3.3 characterizes containment of *sum*-queries, since *sum* is an expandable function. We present a simpler condition for the case where the *sum*-queries are conjunctive. A *sum*-query of the form $q(\bar{x}, sum(y)) \leftarrow \varphi(\bar{z}, \bar{x}, y)$ is associated with a *count*-query $q_c$ defined as $q_c(\bar{x}, y, count) \leftarrow \varphi(\bar{z}, \bar{x}, y)$.

**Theorem 3.12 (Conjunctive Sum-Queries).** *Let $q$ and $q'$ be conjunctive sum-queries. Then $q \subseteq q'$ if and only if $q_c \subseteq q'_c$, and $q \subseteq_b q'$ if and only if $q_c \subseteq_b q'_c$.*

*Proof. (Sketch)* This follows from the the fact that $q \equiv q'$ if and only if $q_c \equiv q'_c$ and $q \equiv_b q'$ if and only if $q_c \equiv_b q'_c$, shown in [18].

## 4 Rewriting Conjunctive Count-Queries

The problem of answering queries using views has been studied for different classes of queries [1,8,11,15]. See [14] for a comprehensive survey of related work. The ability to answer queries using views is useful for query optimization, for maintaining independence of physical and logical database schemas and in the context of data integration. We focus here on finding *maximally-contained sets of rewritings*, a problem that arises when integrating data from varied sources.

We present a sound and complete method to compute maximally-contained sets of rewritings for conjunctive *count*-queries. Our results are also immediately applicable to non-aggregate queries evaluated under bag-set semantics. A similar method can be used to compute maximally-contained sets of rewritings for *sum*-queries. We do not show how to find such sets because of space limitations.

Rewriting aggregate queries has been considered in [6,21,9]. However, to the best of our knowledge, the problem of finding maximally-contained sets of rewritings of aggregate queries has not been dealt with at all. In fact, in [14], this problem is mentioned as an interesting open problem.

### 4.1 Definitions

Let $V$ be a set of queries, called *views*. The queries in $V$ are defined over the relational vocabulary $P$ and use the constants in $C$. We say that a predicate $v$ is *defined* in $V$ if there is a view in $V$ which has the predicate $v$ as its head. We also call $v$ a *view predicate*.

We now allow queries to be defined over the relational vocabulary $V$. Let $r$ be such a query. Any view predicate $v$ in $r$'s body is considered an IDB predicate. Thus, the extension of $v$ with respect to a database $\mathcal{D}$ contains exactly the tuples derived by evaluating $v$ on $\mathcal{D}$, i.e., $v^{\mathcal{D}}$. We denote the result of evaluating $r$ on $\mathcal{D}$, using the view definitions in $V$, as $r^{\mathcal{D}_V}$.

A query that is defined over $P$ is said to be defined over the *base predicates*. A query that is defined over $V$ is said to be defined over the *view predicates*. Let $q$ be a query defined over $P$ and let $r$, $r'$ be queries defined over $V$. We say that

$r$ is *contained modulo* $V$ in $q$, denoted $r \subseteq_V q$, if for all database $\mathcal{D}$, $r^{\mathcal{D}_V} \subseteq q^{\mathcal{D}}$. Similarly, $r \subseteq_V r'$ if $r^{\mathcal{D}_V} \subseteq r'^{\mathcal{D}_V}$. We define *equivalence modulo* $V$, denoted $\equiv_V$, in a similar fashion. If $r \subseteq_V q$ we say that $r$ is a *rewriting* of $q$ using $V$. Note that our notion of a rewriting differs from the standard definition in that $r$ must be contained in $q$ and need not be equivalent to $q$.

Let $q$ be a query, let $\mathcal{R}$ be a set of rewritings of $q$ using views $V$ and let $R$ be a subset of $\mathcal{R}$. We say that $R$ is a *maximally-contained set of rewritings* of $q$ with respect to $\mathcal{R}$ if for every $r \in \mathcal{R}$ there is an $r' \in R$ such that $r \subseteq_V r'$.

### 4.2 Class of Views and Class of Rewritings

We consider the problem of finding rewritings of a conjunctive *count*-query given a set of views. A count-query is sensitive to multiplicities, and count-views are the only type of aggregate views that do not lose multiplicities.[3] Thus, we only use count-views when rewriting count-queries. Even if we restrict ourselves to using *count* views, there may be an infinite number of aggregate terms that can be usable in the head of a rewriting of a count-query. However, we will restrict ourselves to a specific class of queries that is both natural and expressive.

Finding a rewriting of a *count*-query $q$ using a set of views $V$ involves two steps:

- **Generate:** Create a query $r$ over the view predicates $V$.
- **Test:** Determine if $r$ is contained in $q$.

Observe that it is might not possible to check directly if $r \subseteq_V q$ since $r$ may not even be a *count*-query and $r$ uses the view predicates and not the base predicates. Therefore, we have no characterizations of containment that can allow us to determine directly if $r \subseteq_V q$. In order to overcome this problem, we will restrict ourselves to choosing $r$ from the class of queries $\mathit{Unf}(V)$, defined below. We will show that if $r \in \mathit{Unf}(V)$, then it is possible to find a *count*-query $r'$ over the base predicates that is equivalent to $r$. Thus, we will be able to check whether $r \subseteq_V q$ by checking whether $r' \subseteq q$, using the characterizations in Theorem 3.10 and Corollary 3.11.

Let $V = \{v_i(\bar{s}_i, \mathit{count}) \leftarrow \varphi_i(\bar{z}_i, \bar{x}_i)\}_{i \in \mathcal{I}}$ be a set of count-views. Let $r$ be a query defined over $V$ of the form

$$r(\bar{s}, \mathit{sum}(\prod_{j=1}^{n} w_j)) \leftarrow v_1(\theta_1(\bar{s}_1), w_1) \wedge \cdots \wedge v_n(\theta_n(\bar{s}_n), w_n)\,. \tag{2}$$

We call $\theta_i$ the *instantiation* of $v_i$. Note that $w_i$ is the variable that replaces the aggregation term *count* in the head of $v_i$. A view in $V$ can appear any number of times in $r$. The *unfolding* of $r$, denoted $r^u$, is derived by

1. replacing each view atom $v_i(\theta_i(\bar{s}_i), w_i)$ in the body of $r$ by $\varphi_i(\theta'_i(\bar{z}_i), \theta_i(\bar{)}x_i)$ where $\theta'_i$ is an injective mapping of $\bar{z}_i$ to unused variables *and*

[3] Although sum-views are sensitive to multiplicities, they lose these values. For example, sum-queries ignore occurrences of zero values.

2. replacing the aggregation function in the head of $r$ with the function *count*.

Thus, $r^u$ has the form

$$r^u(\bar{s}, count) \leftarrow \varphi_1(\theta'_1(\bar{z}_1), \theta_1(\bar{x}_1)) \wedge \cdots \wedge \varphi_n(\theta'_n(\bar{z}_n), \theta_n(\bar{x}_n))$$

We say that $r$ has the *unfolding property* if for all databases $\mathcal{D}$ it holds that $(r^u)^{\mathcal{D}} \equiv_V r^{\mathcal{D}_V}$. Intuitively, this property states that evaluating $r$ over $\mathcal{D}$ by taking $V$ into consideration will yield the same result as evaluating the unfolding of $r$ on the relations in the database. This is a natural property that is required when rewriting non-aggregate queries. In [6] it has been shown that for a query to have the unfolding property it must be of the form presented in Equation 2. We denote as $Unf(V)$ the class of queries with the form presented in Equation 2 such that $v_i \in V$, for all $i$.

Now, given a *count*-query $q$ and a query $r$ in $Unf(V)$, it is possible to determine if $r \subseteq_V q$, since it is possible to find a *count*-query over the base predicates that is equivalent to $r$. This *count*-query is simply the unfolding of $r$, i.e., $r^u$.

*Example 4.1.* Consider the query $q$ and the views $V = \{v_1, v_2, v_3\}$ defined below:

$$\begin{aligned} q(p, g, count) &\leftarrow \texttt{study}(s, c, g) \wedge \texttt{teach}(p, c) \\ v_1(p, c, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{full\_time}(p) \\ v_2(s, c, g, count) &\leftarrow \texttt{study}(s, c, g) \\ v_3(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{visitor}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

Query $q$ computes the number of students with a given grade, for a given professor. View $v_1$ returns information about the courses that full-time professors teach. (The *count* value will always be one in this view since all the variables in the view are free.) View $v_2$ returns students with courses that they studied and their respective grades. The view $v_3$ computes the number of students with a given grade, for a given visiting professor.

Consider the query $r_1$ defined over the view predicates and its unfolding $r_1^u$

$$\begin{aligned} r_1(p, g, sum(w_1 \times w_2)) &\leftarrow v_1(p, c, w_1) \wedge v_2(s, c, g, w_2) \\ r_1^u(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{full\_time}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

By applying Theorem 3.10 we derive that $r_1^u \subseteq q$ and therefore, $r_1 \subseteq_V q$.

Consider also the query $r_2$ defined over the views and its unfolding $r_2^u$

$$\begin{aligned} r_2(p, g, w_3) &\leftarrow v_3(p, g, w_3) \\ r_2^u(p, g, count) &\leftarrow \texttt{teach}(p, c) \wedge \texttt{visitor}(p) \wedge \texttt{study}(s, c, g)\,. \end{aligned}$$

Once again, using Theorem 3.10, we can verify that $r_2^u \subseteq q$ and thus, $r_2 \subseteq_V q$.

In this manner, we found two rewritings of $q$ using the views. It is possible to show that $\{r_1, r_2\}$ is a maximally-contained set of rewritings of $q$ w.r.t. $Unf(V)$.

However, the following example will show that if only queries from $Unf(V)$ are considered, it may not be possible to find a finite maximally-contained set of rewritings for a given query.

*Example 4.2.* Consider the following query and views

$$q(x, count) \leftarrow a(x, z) \wedge c(x, u)$$
$$v_1(x, y, count) \leftarrow a(x, z) \wedge b(x, y)$$
$$v_2(x, count) \leftarrow c(x, u).$$

The query $r_1$ has the unfolding property (i.e., $r_1 \in \mathit{Unf}(V)$), but is not a rewriting of $q$:

$$r_1(x, sum(w_1 \times w_2)) \leftarrow v_1(x, y, w_1) \wedge v_2(x, w_2).$$

Intuitively, $r_1$ is not a rewriting of $q$ since the atom $b(x, y)$ is bound in $r_1^u$ and does not appear in $q$. However, for any constant $d$, the query $r_d \in \mathit{Unf}(V)$

$$r_d(x, sum(w_1 \times w_2)) \leftarrow v_1(x, d, w_1) \wedge v_2(x, w_2)$$

is a rewriting of $q$. Therefore, if we consider only queries with the unfolding property, a maximally-contained set of rewritings w.r.t. $\mathit{Unf}(V)$ will contain an infinite number of queries (one for each constant).

In order to ensure the existence of a finite maximally-contained set of rewritings, we will choose our rewritings from a class that is more expressive than $\mathit{Unf}(V)$, called $\mathit{PUnf}(V)$. Intuitively, $\mathit{PUnf}(V)$ contains projections of queries in $\mathit{Unf}(V)$. Formally, let $p(\bar{s}, sum(\prod_{j=1}^{n} w_j))$ be a query in $\mathit{Unf}(V)$. We say that the query $r$

$$r(\bar{s}', w) \leftarrow p(\bar{s}, w) \tag{3}$$

is a *k-projection* of $p$ if

1. $\bar{s}'$ contains exactly the first $k$ terms in $\bar{s}$, i.e., $\bar{s} = (\bar{s}', \bar{t})$ for some tuple of terms $\bar{t}$ *and*
2. every variable in $\bar{s}$ and not in $\bar{s}'$ appears only in free atoms in $p^u$.

If $r$ is a $k$-projection of $p$, we say that $p$ is the *k-projectee* of $r$. When evaluating a query such as $r$, the projection is taken as a *set-projection*, i.e., duplicates are removed. Property 2 ensures that each tuple in a $k$-projection is associated with a single aggregation value (see [5]).

Given a set of views $V$, the class $\mathit{PUnf}(V)$ contains all $k$-projections of queries in $\mathit{Unf}(V)$, for all $k$. The reader will note that any query in $\mathit{Unf}(V)$ can be expressed in $\mathit{PUnf}(V)$ by simply taking $\bar{s}'$ as $\bar{s}$.

*Example 4.3.* Recall the query $q$ and views $v_1$ and $v_2$ from Example 4.2. Let $p \in \mathit{Unf}(V)$ and $r \in \mathit{PUnf}(V)$ be the queries

$$p(x, y, sum(w_1 \times w_2)) \leftarrow v_1(x, y, w_1) \wedge v_2(x, w_2)$$
$$r(x, w) \leftarrow p(x, y, w)\,.$$

It is not difficult to see that $r \subseteq_V q$ and for all $d$, it holds that $r_d \subseteq_V r$. (We show formally how to check for containment below.) In fact, one can show that the set $\{r\}$ is a maximally contained rewriting of $q$ w.r.t. $\mathit{PUnf}(V)$.

Given a query $q$ and a set of views $V$, we will be interested in finding a maximally-contained set of rewritings of $q$ w.r.t. $PUnf(V)$. We extend the definition of a containment mapping (page 118) to allow us to check if a query in $PUnf(V)$ is contained in a *count*-query. Consider the queries $p(\bar{s}, count)$ and $p'(\bar{s}', count)$. Suppose that the tuples $\bar{s}$ and $\bar{s}'$ are at least of size $k$. We say that a mapping $\theta$ from the variables in $p'$ to the terms in $p$ is a $k$-containment mapping if it fulfills requirements 2 through 4 of containment mappings and also maps the first $k$ terms in $\bar{s}'$ to the first $k$ terms in $\bar{s}$. Our $k$-containment mappings can be used to check if a query in $PUnf(V)$ is contained in a *count*-query.

**Lemma 4.4 (Containment of $PUnf(V)$ Queries).** *Let $q$ be a conjunctive count-query, $V$ be a set of views and $r$ be a query in $PUnf(V)$. Suppose that $p$ is the $k$-projectee of $r$. Then $r$ is a rewriting of $q$, i.e., $r \subseteq_V q$, if and only if there exists a $k$-containment mapping from $q$ to $p^u$.*

Lemma 4.4 implies an algorithm for checking if a query in $PUnf(V)$ is a rewriting. Hence, when searching for a maximally-contained set of rewritings, we simply have to generate queries in $PUnf(V)$ and check if they are rewritings. For this procedure to terminate, we show that it is sufficient to generate a finite number of queries. In fact, it is possible to bound the number of views being used and the number of constants needed. This is similar to the conjunctive case. Using Lemma 4.4 and 4.5, we can show that it is possible to find a maximally-contained set of rewritings of a given query w.r.t. the class $PUnf(V)$.

**Lemma 4.5 (Size of Rewriting).** *Let $q$ be a query and let $r \in PUnf(V)$ be a rewriting of $q$. Suppose that the body of $q$ contains $n$ atoms. Then there is a rewriting $r'$ of $q$ such that $r \subseteq_V r'$ and the projectee of $r'$*

- *contains at most $n$ views* and
- *does not contain any constant not appearing in either $V$ or $q$.*

**Theorem 4.6 (Maximally-Contained Set of Rewritings).** *Let $q$ be a query and $V$ be a set of views. It is possible to find a maximally-contained set of rewritings of $q$ w.r.t. $PUnf(V)$ of finite size.*

Theorem 4.6 suggests an exponential algorithm for finding a maximally-contained set of rewritings. It can be improved using standard rewriting algorithm techniques, such as those appearing in [7,19]. Such algorithms have been shown to run well in practice [19].

## 5 Conclusion

The class of expandable aggregation functions has been introduced and we have shown how to reduce containment of queries with expandable aggregation functions to equivalence of queries. This reduction holds even in the presence of arbitrary integrity constraints. Simpler characterizations for containment of

*count* and *sum*-queries have been presented. The problem of finding maximally-contained sets of rewritings has been solved for conjunctive *count*-queries.

Containment of non-aggregate conjunctive queries under bag and bag-set semantics has been studied in [3,12]. In these papers, a query $q$ is contained in a query $q'$ if for all databases $\mathcal{D}$, it holds that $q^{\mathcal{D}}$ is a *sub-bag* of $q'^{\mathcal{D}}$. For *count*-queries, this is similar to defining that $q(\bar{x}, count)$ is contained in $q'(\bar{x}, count)$ if for all databases $\mathcal{D}$ and tuples $\bar{d}$, if $(\bar{d}, c) \in q^{\mathcal{D}}$, then there is a $c' \geq c$ such that $(\bar{d}, c') \in q'^{\mathcal{D}}$. It has been shown independently by [3] and by [12] that containment of conjunctive queries under bag semantics is decidable if the queries do not contain any repeated predicates. Undecidability of containment of unions of conjunctive queries under bag semantics has been proven in [12].

Our definition of containment differs from the definition above and we feel that it is more natural in the context of finding maximally-contained sets of rewritings. For our definition of containment, we know that if $q \subseteq q'$ then every result of applying $q$ on a database would also be achieved by $q'$. The corresponding definition of containment in [3,12] would only allow us to derive lower bounds on the *count* values of $q'$. Moreover, our definition carries over easily to other types of aggregation functions.

We leave for future research the problem of deciding containment of queries with a `HAVING` clause, as well as queries containing aggregation functions that are not expandable, such as *prod* and *parity*. Finding tight upper and lower bounds for the complexity of containment is another important open problem.

## References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In J. Paredaens, editor, *Proc. 17th Symposium on Principles of Database Systems*, pages 254–263, Seattle (Washington, USA), June 1998. ACM Press.

[2] S. Chaudhuri, S. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In P.S.Yu and A. Chen, editors, *Proc. 11th International Conference on Data Engineering*, Taipei, Mar. 1995. IEEE Computer Society.

[3] S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. 12th Symposium on Principles of Database Systems*, Washington (D.C., USA), May 1993. ACM Press.

[4] S. Cohen, W. Nutt, and Y. Sagiv. Equivalences among aggregate queries with negation. In *Proc. 20th Symposium on Principles of Database Systems*, pages 215–226, Santa Barbara (California, USA), May 2001. ACM Press.

[5] S. Cohen, W. Nutt, and Y. Sagiv. Containment of aggregate queries (extended version), Oct. 2002. Available at `http://www.cs.huji.ac.il/~sarina/papers/agg-containment-long.ps`.

[6] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, pages 155–166, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[7] S. Cohen, W. Nutt, and A. Serebrenik. Algorithms for rewriting aggregate queries using views. In *Symposium on Advances in Databases and Information Systems, Enlarged Fourth East-European Conference on Advances in Databases and Information Systems*, pages 65–78, Prague (Czech Republik), Sept. 2000.

[8] O. Duschka and M. Genesereth. Answering recursive queries using views. In *Proc. 16th Symposium on Principles of Database Systems*, pages 254–263, Tucson (Arizona, USA), May 1997. ACM Press.

[9] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In C. Papadimitriou, editor, *Proc. 18th Symposium on Principles of Database Systems*, pages 174–183, Philadelphia (Pennsylvania, USA), May 1999. ACM Press.

[10] S. Grumbach and L. Tininini. On the content of materialized aggregate views. In *Proc. 19th Symposium on Principles of Database Systems*. ACM Press, 2000.

[11] A. Gupta and I. S. Mumick, editors. *Materialized Views—Techniques, Implementations and Applications*. MIT Press, 1999.

[12] Y. Ioannidis and R. Ramakrishnan. Beyond relations as sets. *ACM Transactions on Database Systems*, 20(3):288–324, 1995.

[13] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, 1996.

[14] A. Levy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[15] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. 14th Symposium on Principles of Database Systems*, pages 95–104, San Jose (California, USA), May 1995. ACM Press.

[16] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. 22nd International Conference on Very Large Data Bases*, pages 251–262, Bombay (India), Sept. 1996.

[17] F. Llirbat, F. Fabret, and E. Simon. Eliminating costly redundant computations from SQL trigger executions. In *Proc. 1997 ACM SIGMOD International Conference on Management of Data*, pages 428–439, Tucson (Arizona, USA), June 1997.

[18] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In J. Paredaens, editor, *Proc. 17th Symposium on Principles of Database Systems*, pages 214–223, Seattle (Washington, USA), June 1998. ACM Press. Long version as Report of Esprit LTR DWQ.

[19] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In P. 26th International Conference on Very Large Data Bases, editor, *Proc. 26th International Conference on Very Large Data Bases*, Cairo (Egypt), 2000. Morgan Kaufmann Publishers.

[20] K. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pages 447–458, Montreal (Canada), June 1996.

[21] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. 22nd International Conference on Very Large Data Bases*, Bombay (India), Sept. 1996. Morgan Kaufmann Publishers.

[22] J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I: Classical Database Systems*. Computer Science Press, New York (New York, USA), 1988.

[23] H. Yang and P.-A. Larson. Query transformation for PSJ queries. In *Proc. 13th International Conference on Very Large Data Bases*, pages 245–254, Brighton (England), Sept. 1987. Morgan Kaufmann Publishers.

# Auditing Sum Queries

Francesco M. Malvestuto[1] and Mauro Mezzini[2]

[1] Dipartimento di Informatica, Università "La Sapienza", Via Salaria 113, 00198 Roma, Italy
[2] Telecom Italia, Roma, Italy

**Abstract.** In an on-line statistical database, the query system should leave unanswered queries asking for sums that could lead to the disclosure of confidential data. To check that, every sum query and previously answered sum queries should be audited. We show that, under a suitable query-overlap restriction, an auditing procedure can be efficiently worked out using flow-network computation.

## 1 Introduction

Statistical databases raise concerns on the compromise of individual privacy, a statistical database [1] being an ordinary database which contains information about individuals (persons, companies, organisations etc.) but its users are only allowed to access sums of individual data provided that they do not lead to the disclosure of confidential data. Consider a relation scheme $R$ = {NAME, SSN, AGE, DEPARTMENT, SALARY} and a statistical-query system which answers only queries such as: "What is the sum of salaries of the individuals qualified by the condition $P$" where $P$ is a "category predicate", that is, a condition on the domain of the pair {AGE, DEPARTMENT} of "category" attributes such as DEPARTMENT $\neq$ Direction & AGE $\geq$ 40. Such a query is called a (*categorical*) *sum query* on SALARY. Assume further that SALARY is a confidential attribute. Given a relation with scheme $R$, if the set $T$ of tuples selected by $P$ is a singleton, say $T = \{t\}$, then the response to the sum query above (if released) would allow the salary of the individual corresponding to the tuple $t$ to be disclosed and, therefore, it should be denied. The response to such a sum query is called a *sensitive sum* [13, 14]. Indeed, more sophisticated sensitivity criteria exist which broaden the class of sum queries that should be left unanswered [13, 14]. A (memoryless) security measure that leaves unanswered only sum queries whose responses are sensitive sums is not adequate for it does not exclude the possibility of computing some sensitive sum by combining nonsensitive responses. What measures suffice to avoid the disclosure of sensitive sums? This is the classical version of the *security control problem* for sum queries. Indeed, a stronger version of this problem is of practical interest since the producers of statistical databases (e.g., Census Bureaux) demand that an "accurate" estimate of no sensitive sum should be possible in the sense that, if $s$ is a sensitive sum, then the possible values of $s$ consistent with the information released by the statistical-query system should span an interval that is not entirely contained in the interval $[(1-p)\,s, (1+p)\,s]$ where $p$ is fixed percentage, called the *protection level* [13, 14]. Note that for $p = 0$, one has the

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 126-142, 2003.

classical version of the security problem: the exact value of no sensitive sum should be disclosed. The program of the statistical query system that should ensure the protection of sensitive sums is called the "auditor" [3, 10, 13] and it should work as follows. Let $R$ be a relation scheme containing a confidential attribute S with domain $\sigma$, and let $K$ be the set of category attributes in $R$. Suppose that sum queries $Q(1)$, …, $Q(n–1)$ on S have been already answered when a new sum query $Q(n)$ on S arrives. Let $P(v)$ and $q(v)$ be the category predicate and the response to $Q(v)$, $1 \le v \le n$. Without loss of generality [11], we assume that each $P(v)$ is of the form "$K$ in $\kappa(v)$" where $\kappa(v)$ is a nonempty subset of the domain of $K$. The amount of information that would be released to the users if $Q(n)$ were answered will represented by a model which consists of a semantic part and an analytic part. The semantic part of the model contains the overlap relationships among the sum queries $Q(v)$ and consists of a hypergraph $G = (V, A)$ where $V = \{1, \ldots, n\}$ and $A$ contains the nonempty subsets $a$ of $V$ such that the subset of the domain of $K$

$$\kappa(a) = \leftrightarrow_{v \in a}\, \kappa(v) \;-\; \approx_{v \notin a}\, \kappa(v)$$

is not empty. The analytic part of the model consists of the system of linear constraints

$$\begin{cases} \sum_{a \in A(v)} x(a) = q(v) & (v \in V) \\ x(a) \in \sigma & (a \in A) \end{cases} \tag{1}$$

where $A(v) = \{a \in A: v \in a\}$. Here, variable $x(a)$ stands for the unknown (to the users) sum of S over the set of tuples selected by the predicate "$K$ in $\kappa(a)$". Thus, a "snooper" can compute the two quantities $\min_X x(a)$ and $\max_X x(a)$ for each hyperarc $a$ of $G$, where $X$ is the solution set of constraint system (1). In order to decide if the response to $Q(n)$ leads to the disclosure of some sensitive sum, the auditor will compute the actual value $s(a)$ of each $x(a)$ and apply the sensitivity criterion in use to decide if $s(a)$ is sensitive; next, for each $a$ having $s(a)$ sensitive, it will apply the safety test which checks that

$$\min_X x(a) < (1–p)\, s(a) \qquad \text{or} \qquad \max_X x(a) > (1+p)\, s(a)$$

where $p$ is the protection level in use. If each sensitive sum $s(a)$ is protected at level $p$, then (and only then) the auditor will decide that $Q(n)$ can be answered safely.

Suppose that the weighted hypergraph $(G, s)$, where $s$ is the function on $A$ that assigns to each hyperarc $a$ of $G$ the corresponding sum $s(a)$, has been constructed and suppose that a number of sensitive weights have been found (for details see [10, 12]). What remains to do is testing each sensitive sum $s(a)$ for safety, which requires computing $\min_X x(a)$ and $\max_X x(a)$. In what follows, the weighted hypergraph $(G, s)$ will be referred to as the *map* of $\{Q(1), \ldots, Q(n)\}$ and the two quantities $\min_X x(a)$ and $\max_X x(a)$ are called the *tightest lower bound* and the *tightest upper bound* on the weight of the hyperarc $a$ of the map $(G, s)$, respectively.

*Example* 1. Consider a relation with scheme {NAME, DEPARTMENT, SALARY}. We make the following assumptions:

— the attribute SALARY is confidential and its domain is the set of nonnegative real numbers,
— the attribute DEPARTMENT is the only category attribute and its domain is {A, B, C, D, E, F},
— the sums of SALARY over the employees in the single departments are as follows: 15.0 for A, 9.0 for B, 7.5 for C, 6.5 for D, 6.0 for E and 5.5 for F,
— the sum of SALARY 15.0 for A is the only sensitive sum,
— the protection level in use is $p = 0$.

Consider four sum queries on SALARY specified by the following category sets: $\kappa(1)$ = {A, B}, $\kappa(2)$ = {A, C, D}, $\kappa(3)$ = {B, C, F} and $\kappa(4)$ = {D, E}. The map of the four sum queries can be pictured as a graph (see Figure 1) whose arcs correspond to $\kappa(1, 2)$ = {A}, $\kappa(1, 3)$ = {B}, $\kappa(2, 3)$ = {C}, $\kappa(2, 4)$ = {D}, $\kappa(3, 3)$ = {F} and $\kappa(4, 4)$ = {E}.

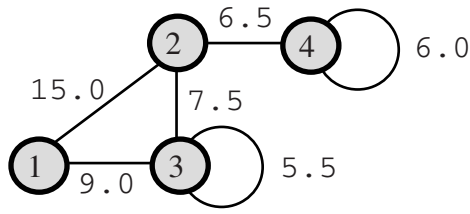



**Fig. 1.**

The only sensitive weight is that of the arc (1, 2). Using standard algebraic methods, one finds that the tightest bounds on its weight are 9.25 and 24. So, the set of the four sum queries is safe. Suppose now that a fifth sum query arrives with category set $\kappa(5)$ = {E, F}. The new query map is shown in Figure 2.

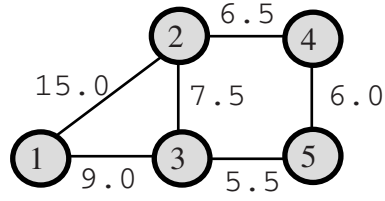



**Fig. 2.**

Now, the tightest bounds on the weight of the arc (1, 2) coincide both with its weight. Consequently, the auditor will leave the fifth query unanswered. ■

What makes the work of the auditor hard is that the number of hyperarcs of the map of $\{Q(1), \ldots, Q(n)\}$ may be exponential in $n$. To overcome this difficulty, a query-

overlap restriction [1] can be introduced which, for a fixed positive integer $r$, requires that the response to the (current) sum query $Q(n)$ is soon denied if in $\{Q(1), \ldots, Q(n)\}$ there is a *nonempty overlap of order r*, that is, if there exists a subset $\{Q(v): v \in V'\}$ of size $r$ such that $\leftrightarrow_{v \in V'} \kappa(v) \neq \emptyset$. The simplest nontrivial case is $r = 2$ [10, 12], which implies that the map of $\{Q(1), \ldots, Q(n)\}$ is a graph (where loops are allowed) as in Example 1.

In this paper we address the problem of computing the tightest bounds on the weight of a graph under the assumption that the arc weights are nonnegative real numbers. This problem can be solved using linear programming methods [5, 6]. A more efficient approach consists in transforming that into a network flow problem and Gusfield [7] proved that, in the bipartite case, the tightest bounds on the weight of an arc can be computed with two maximum-flow computations. Here we shall show that for a nonbipartite graph the tightest bounds on the weight of an arc can be found with two or four maximum-flow computations depending on whether the arc is or is not a loop.

The paper is organised as follows. In Section 2 we review the results on bipartite maps. In Section 3 we present a bipartite transformation of a nonbipartite map. In Section 4 we present the maximum-flow computations for the tightest bounds on the weight of an arc of a nonbipartite map. Section 5 deals with a special case where there are closed formulas for the tightest bounds on the weight of each arc. Section 6 contains some open problems.

## 2 The Bipartite Case

In this section, we review the maximum-flow technique proposed by Gusfield [7] to compute the tightest bounds on the weight of a given arc of a map $(G, s)$ where $G = (V, A)$ is a bipartite, connected graph. Let $\{V_1, V_2\}$ be the vertex bipartition of $G$. First, a flow network [2] is built up as follows. Let $q(v) = \sum_{a \in A(v)} s(a)$ for each $v$ in $V$, and let $M$ be a finite number larger than $max\ \{q(v): v \in V_1\}$. First, each arc $(u, v)$ of $G$ is directed both from $V_1$ to $V_2$ and from $V_2$ to $V_1$. Then, the capacity of each arc $u \varnothing v$ is set to $M$ if $u \in V_1$, and to $s(u, v)$ otherwise. We denote the resulting flow network by $N(G, s; V_1, M)$.

*Example* 2. Consider the map $(G, s)$ shown in Figure 3.

Let $V_1 = \{2, 4, 6\}$ and $V_2 = \{1, 3, 5\}$, and let $M = 2$. Figure 4 shows the network $(G, s; V_1, M)$.

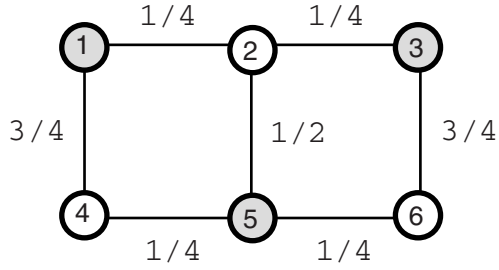



**Fig. 3.** A bipartite map

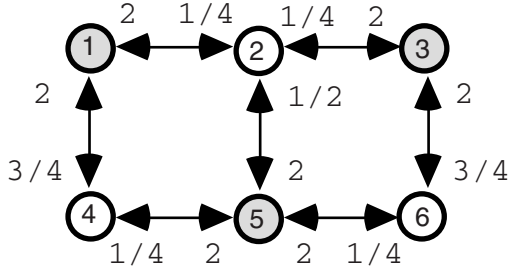



**Fig. 4.** The network associated with the map of Figure 3 ■

Let $f_{u,v}$ be a maximum flow in $N(G, s; V_1, M)$ from vertex $u$ to vertex $v$, and let $F_{u,v}$ be the value of $f_{u,v}$, that is,

$$F_{u,v} = \sum_w f_{u,v}(u \varnothing w) - \sum_w f_{u,v}(w \varnothing u) = \sum_w f_{u,v}(w \varnothing v) - \sum_w f_{u,v}(v \varnothing w)$$

Note that if $u \in V_1$ then

$$F_{u,v} = M + \sum_{w \in V_2 - \{v\}} f_{u,v}(u \varnothing w) - \sum_{w \in V_2 - \{v\}} f_{u,v}(w \varnothing u) \ .$$

The following three propositions were proved by Gusfield [7]. Recall that $X$ is the set of solutions of constraint system (1) where $\sigma$ is the set of nonnegative reals.

**Proposition 1.** Given a map $(G, s)$ where $G$ is a bipartite connected graph with bipartition $\{V_1, V_2\}$, let $a$ be an arc of $G$ with end-points $u \in V_1$ and $v \in V_2$. Then

$$\min_X \ x(a) = max\ \{0, s(a) + M - F_{u,v}\} \qquad \text{and} \qquad \max_X \ x(a) = F_{v,u}$$

where $F_{u,v}$ and $F_{v,u}$ are the values of maximum flows in $N(G, s; V_1, M)$ from $u$ to $v$ and from $v$ to $u$, respectively.

If $x \in X$ has $x(a) = \min\limits_{X} x(a)$ ($x(a) = \max\limits_{X} x(a)$, respectively), we call the map $(G, x)$ an *a-minimal* (*a-maximal*, respectively) *variant* of the map $(G, s)$.

**Proposition 2.** Given a map $(G, s)$ where $G$ is a bipartite connected graph with bipartition $\{V_1, V_2\}$, let $a$ be an arc of $G$ with end-points $u \in V_1$ and $v \in V_2$.

(*i*) Given a maximum flow $f_{u,v}$ in $N(G, s; V_1, M)$ from $u$ to $v$, an $a$-minimal variant $(G, x)$ of $(G, s)$ can be constructed as follows. For each arc $a'$ of $G$ with end-points $u' \in V_1$ and $v' \in V_2$ take

$$x(a') = \begin{cases} max\{0, s(a) + M - F_{u,v}\} & \text{if } a' = a \\ s(a') + \tau[f_{u,v}(u' \to v') - f_{u,v}(v' \to u')] & \text{else} \end{cases}$$

where $\tau = min\left\{1, \dfrac{s(a)}{F_{u,v} - M}\right\}$.

(*ii*) Given a maximum flow $f_{v,u}$ in $N(G, V_1, V_2; s, M)$ from $v$ to $u$, an $a$-maximal variant $(G, x)$ of $(G, s)$ can be constructed as follows. For each arc $a'$ of $G$ with end-points $u' \in V_1$ and $v' \in V_2$ take

$$x(a') = \begin{cases} F_{v,u} & \text{if } a' = a \\ s(a') + f_{v,u}(u' \to v') - f_{v,u}(v' \to u') & \text{else} \end{cases}$$

*Example* 2 (continued). Figure 5(*a*) shows a maximum flow $f_{2,5}$ from the vertex 2 to the vertex 5 and Figure 5(*b*) shows a maximum flow $f_{5,2}$ from the vertex 5 to the vertex 2. So, $F_{2,5} = 7/2$ and $F_{5,2} = 1$.

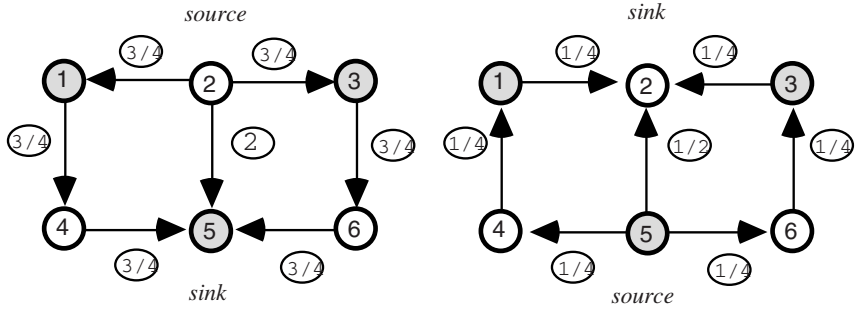



**Fig. 5.**

By Proposition 1, the tightest lower and upper bounds on the weight of the arc (2, 5) are 0 and 1, respectively. Using Proposition 2, we obtain a (2, 5)-minimal variant and a (2, 5)-maximal variant of the map of Figure 3, which are shown in Figures 6(*a*) and 6(*b*) respectively.

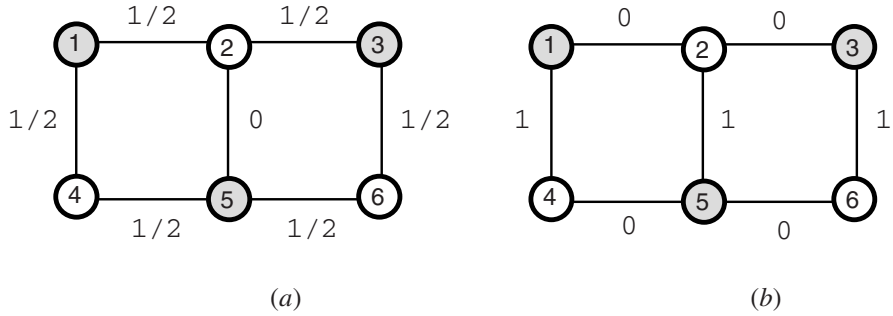



**Fig. 6.** Minimal and maximal variants ■

**Proposition 3.** Given a map $(G, s)$ where $G$ is a complete bipartite graph with bipartition $\{V_1, V_2\}$, let $q(v) = \sum_{a \in A(v)} s(a)$, $v \in V$, and let $N = \sum_{v \in V_1} q(v)$. For each arc $a$ of $G$ with end-points $u$ and $v$, one has

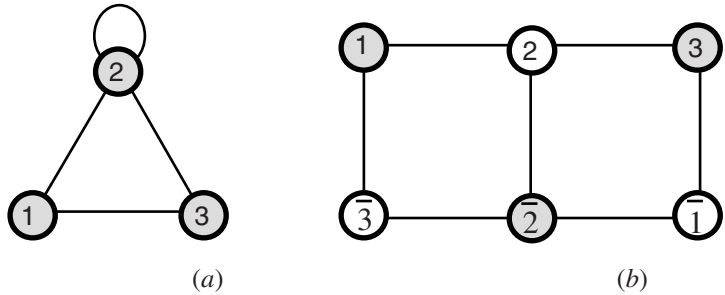
$$\min_X x(a) = \max\{0, q(u) + q(v) - N\}$$
$$\max_X x(a) = \min\{q(u), q(v)\}.$$

## 3 A Bipartite Map Associate with a Nonbipartite Map

Consider now a nonbipartite, connected graph $G = (V, A)$. The arcs of $G$ that are not loops will be referred to as *links*. With $G$ we associate a bipartite, connected graph $H = (W, B)$ with $|W| = 2|V|$ and $|B| = 2|A| - l$, where $l$ is the number of loops of $G$. The graph $H$, which will be referred to as a *bipartite transform* of $G$ [9], is constructed as follows. Let $\overline{V} = \{\overline{v} : v \in V\}$ be a "copy" of $V$ (that is, $\overline{V} \leftrightarrow V = \emptyset$). The vertex set of $H$ is taken to be $W = V \approx \overline{V}$. The arc set $B$ of $H$ is defined as follows. Arbitrarily chosen a spanning tree $T$ of $G$, let $G'$ be the bipartite graph obtained from $T$ by adding all nontree arcs that do not create odd cycles. Take $B = \approx_{a \in A} \beta(a)$ where $\beta$ is the function defined on $A$ as follows:

$$\beta(u, v) = \begin{cases} \{(u,v),(\overline{u},\overline{v})\} & \text{if } (u,v) \text{ is an edge of } G' \\ \{(u,\overline{v}),(\overline{u},v)\} & \text{if } (u,v) \text{ is a link of } G \text{ but not of } G' \\ \{(u,\overline{u})\} & \text{if } u = v \text{ and } (u,u) \text{ is a loop of } G \end{cases}$$

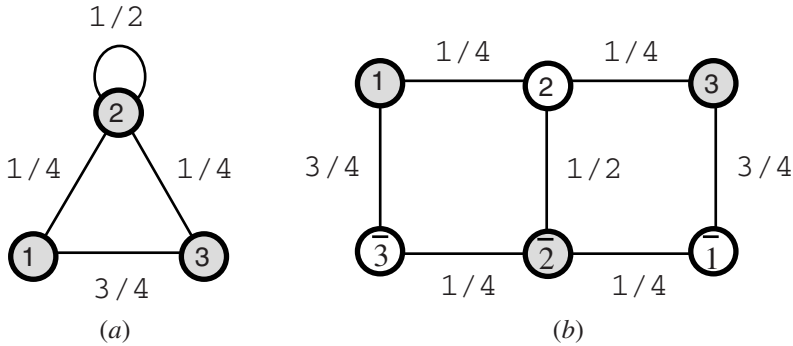
For vertex $w$ of $H$, if $w = v$ or $w = \overline{v}$, we say that $v$ is the vertex of $G$ corresponding to $w$; similarly, for arc $b$ of $H$, if $b$ belongs to $\beta(a)$, we say that $a$ is the arc of $G$ corresponding to $b$. Let $\{V_1, V_2\}$ be the bipartition of the vertex set of $G'$ and let $\overline{V}_i = \{\overline{v} : v \in V_i\}$, $i = 1, 2$. Note that $H$ is a bipartite, connected graph with bipartition $\{W_1, W_2\}$ where $W_1 = V_1 \approx \overline{V}_2$ and $W_2 = \overline{V}_1 \approx V_2$.




($a$) ($b$)

**Fig. 7.** A nonbipartite graph and a bipartite transform ■

Given a nonbipartite, connected map $(G, s)$, let $H = (W, B)$ be a bipartite transform of $G$ and let $t$ be the function defined on $B$ with $t(b) = s(a)$ if $a$ is the arc of $G$ corresponding to $b$. We call $(H, t)$ a bipartite map *associated* with $(G, s)$.




($a$) ($b$)

**Fig. 8.** ($a$) A nonbipartite map; ($b$) an associated bipartite map ■

For vertex $w$ of $H$, let $r(w) = \sum_{b \in B(w)} t(b)$ where $B(w)$ is the set of arcs incident to $w$. Let $Y$ be the set of nonnegative real-valued solutions of the equation system

$$\sum_{b \in B(w)} y(b) = r(w) \qquad (w \in W) \qquad (2)$$

The following two obvious facts show that $Y$ is closely connected with $X$.

**Fact 1**. For every $x$ in $X$, the function $y$ on $B$ with $y(b) = x(a)$, where $a$ is the arc of $G$ corresponding to $b$, belongs to $Y$.

**Fact 2**. For every $y$ in $Y$, the function $x$ on $A$ with

$$x(a) = \begin{cases} y(b) & \text{if } a \text{ is a loop of } G \text{ with } \beta(a) = \{b\} \\ \frac{1}{2}[y(b') + y(b'')] & \text{if } a \text{ is a link of } G \text{ with } \beta(a) = \{b', b''\} \end{cases}$$

belongs to $X$.

## 4 Bounds on the Weight of an Arc

In this section we show that the tightest bounds on the weight of an arc of a nonbipartite map can be computed with two or four maximum-flow computations depending whether the arc is a loop or a link. The following is an immediate consequence of Facts 1 and 2.

**Lemma 1**. Let $(G, s)$ be a nonbipartite map, and let $(H, t)$ be a bipartite map associated with $(G, s)$. For each arc $a$ of $G$,

($i$) if $a$ is a loop with $\beta(a) = \{b\}$, then the tightest lower and upper bounds on the weight of $a$ coincide with the tightest lower and upper bounds on the weight of $b$, respectively;

($ii$) if $a$ is a link with $\beta(a) = \{b', b''\}$, then the tightest lower and upper bounds on the weight of $a$ are respectively equal to

$$\frac{1}{2} \min_{Y} [y(b') + y(b'')] \quad \text{and} \quad \frac{1}{2} \max_{Y} [y(b') + y(b'')].$$

By part ($i$) of Lemma 1, the tightest bounds on the weight of a loop of $(G, s)$ coincide with the tightest bounds on the weight of the corresponding arc of $(H, t)$ and, hence, by Proposition 1 they can be determined with two maximum-flow computations.

*Example* 3. Consider the nonbipartite map of Figure 8($a$). We apply Part ($i$) of Lemma 1 to compute the tightest bounds on the weight of the loop (2, 2). The associated bipartite map of Figure 8($b$) looks like the map of Figure 3. Therefore, the tightest bounds on the weight of the loop (2, 2) of the map of Figure 8($a$) coincide with the tightest bounds on the weight of the arc (2, 5) of the map of Figure 3, which were `0` and `1`. ■

Consider now the case of a link $a$ of $(G, s)$ with $\beta(a) = \{b', b''\}$. By Part ($ii$) of Lemma 1, we have to compute

$$\min_{Y} [y(b') + y(b'')] \quad \text{and} \quad \max_{Y} [y(b') + y(b'')].$$

We shall show that they can be obtained as follows: The minimum (maximum, respectively) of the function $y(b') + y(b'')$ over $Y$ equals the tightest lower (upper, respectively) bound on the weight of the arc $b'$ of $H$ plus the tightest lower (upper, respectively) bound on the weight of the arc $b''$ of the map that is obtained from a $b'$-minimal ($b'$-maximal, respectively) variant of $(H, t)$ by deleting $b'$.

We now prove the correctness for $\max_Y [y(b') + y(b'')]$. The proof of the correctness for $\min_Y [y(b') + y(b'')]$ is similar.

Consider the bipartite graph $H = (W, B)$. A *circulation* in $H$ is any real-valued solution $z$ of the homogeneous equation system associated with equation system (2); that is,

$$\sum_{b \in B(w)} z(b) = 0 \qquad (w \in W)$$

The *signed support* of circulation $z$ is the ordered couple $(S^+, S^-)$ where $S^+ = \{b \in B: z(b) > 0\}$ and $S^- = \{b \in B: z(b) < 0\}$, and the *support* of $z$ is the set $S^+ \approx S^-$. A nonzero circulation $z$ in $H$ is *minimal* if there is no circulation whose support is a proper subset of the support of $z$. It is well-known from linear algebra that, for every (nonzero) circulation $z$ and for each $b$ with $z(b) \neq 0$, there exists a minimal circulation $\zeta$ such that $b$ belongs to the support of $\zeta$ and, if $(S^+, S^-)$ and $(C^+, C^-)$ are respectively the signed supports of $z$ and $\zeta$, then $C^+ \prod S^+$ and $C^- \prod S^-$. Finally, since $H$ is a bipartite graph, the supports of minimal circulations are all and the only cycles (i.e., simple circuits) of $H$ (e.g., see [4]). To sum up, one has the following

**Fact 3**. Let $z$ be a circulation in $H$ with signed support $(S^+, S^-)$, and let $b$ be an element of the support of $z$. Then $b$ lies in a simple cycle $C$ which is the support of a minimal circulation in $H$ having $(C \leftrightarrow S^+, C \leftrightarrow S^-)$ as its signed support.

**Lemma 2**. Let $(H, t)$ be a bipartite map and let $b'$ and $b''$ be two arcs of $H$. There exists a nonnegative real-valued solution $y$ of equation system (2) that maximises both $y(b')$ and $y(b') + y(b'')$.

*Proof.* Let $(H, y_o)$ be a $b'$-maximal variant of $(H, t)$. Moreover, let $Y_1$ be the set of nonnegative real-valued solutions of equation system (2) that maximise $y(b') + y(b'')$, and let $y_1$ be in $Y_1$ and such that $y_1(b') \geq y(b')$ for every $y \in Y_1$. Of course, one has $y_1(b') \leq y_o(b')$. We shall prove that $y_1(b') = y_o(b')$. Consider the circulation $z = y_o - y_1$, and let $(S^+, S^-)$ be the signed support of $z$. Suppose, by contradiction, that $y_1(b') < y_o(b')$. Then $b'$ belongs to $S^+$. By Fact 3, $b'$ lies in a cycle $C$ which is the support of a minimal circulation having signed support $(C^+, C^-)$, where $C^+ = C \leftrightarrow S^+$ and $C^- = C \leftrightarrow S^-$. Such a minimal circulation can be explicitly constructed as follows. Let

$$\varepsilon = \min \{|c(b)|: b \in C^-\}$$

and let $\zeta$ be the circulation with

$$\zeta(b) = \begin{cases} +\varepsilon & b \in C^+ \\ -\varepsilon & b \in C^- \\ 0 & \text{else} \end{cases}$$

Let $y_2 = y_1 + \zeta$. Of course, $y_2$ is a solution of equation system (2). Indeed, $y_2$ is nonnnegative everywhere because, for each arc $b$ of $H$, if $b$ is not in $C^-$, then

$$y_2(b) = y_1(b) + \zeta(b) \geq y_1(b) \geq 0;$$

otherwise,

$$y_2(b) = y_1(b) - \varepsilon \geq y_1(b) + z(b) = y_o(b) \geq 0 .$$

We now show that from the foregoing a contradiction always follows. Consider the following three cases that can occur for $b''$:

Case 1. $b''$ is in $C^+$. Then $y_2(b'') = y_1(b'') + \varepsilon$ which leads to the following

$$y_2(b') + y_2(b'') = y_1(b') + y_1(b'') + 2\varepsilon > y_1(b') + y_1(b'')$$

which contradicts the membership of $y_1$ in $Y_1$.

Case 2. $b''$ is in $C^-$. Then $y_2(b'') = y_1(b'') - \varepsilon$ and, hence, one has

$$y_2(b') + y_2(b'') = y_1(b') + \varepsilon + y_1(b'') - \varepsilon = y_1(b') + y_1(b'')$$

so that $y_2$ belongs to $Y_1$. But, since $b'$ is in $C^+$, one has

$$y_2(b') = y_1(b') + \zeta(b') = y_1(b') + \varepsilon > y_1(b')$$

which contradicts the choice of $y_1$.

Case 3. $b'' \in C^+ \approx C^-$. Then $y_2(b'') = y_1(b'')$ which leads to the following

$$y_2(b') + y_2(b'') = y_1(b') + \varepsilon + y_1(b'') > y_1(b') + y_1(b'')$$

which contradicts the membership of $y_1$ in $Y_1$. ✹

The following is an immediate consequence of Lemma 2.

**Lemma 3**. The maximum of the function $y(b') + y(b'')$ over the set $Y$ of nonnegative real-valued solutions of equation system (2) equals the tightest upper bound on the weight of the arc $b'$ of $(H, t)$ plus the tightest upper bound on the weight of the arc $b''$ of the map obtained from a $b'$-maximal variant of $(H, t)$ by deleting $b'$.

Combining Lemmas 1 and 3 with Proposition 1, we obtain the following algorithm which, given a bipartite map $(H, t)$ of $(G, s)$ and a link $a$ of $G$ with $\beta(a) = \{b', b''\}$, computes the tightest upper bound on the weight of a link $a$ of a nonbipartite map $(G, s)$. The input data are:

| | |
|---|---|
| $(H, t)$ | a bipartite map associated with $(G, s)$ |
| $\{W_1, W_2\}$ | the bipartition of $H$ |
| $N(H, t; W_1, M)$ | a network associated with $(H, t)$ |

$b' = (w_1', w_2')$ with $w_1' \in W_1$ and $w_2' \in W_2$ , $b'' = (w_1'', w_2'')$ with $w_1'' \in W_1$ and $w_2'' \in W_2$.

Algorithm MAX

(1) Find a maximum flow $f$ in $N(H, t; W_1, M)$ from $w_2'$ to $w_1'$, and let $F$ be the value of $f$.

(2) Given $f$ and using Proposition 2(*ii*), construct a $b'$-maximal variant $(H, y)$ of $(H, t)$.

(3) Let $(H', t')$ be the map obtained from $(H, y)$ by deleting $b'$. Find a maximum flow $f'$ in $N(H', t'; W_1, M)$ from $w_2''$ to $w_1''$, and let $F'$ be the value of $f'$.

(4) Set the tightest upper bound on the weight of $a$ to $\dfrac{F + F'}{2}$.

Analogously, the following algorithm correctly computes the tightest lower bound on the weight of $a$.

Algorithm MIN

(1) Find a maximum flow $f$ in $N(H, t; W_1, M)$ from $w_1'$ to $w_2'$.

(2) Given $f$ and using Proposition 2(*i*), construct a $b'$-minimal variant $(H, y)$ of $(H, t)$.

(3) Let $(H', t')$ be the map obtained from $(H, y)$ by deleting $b'$. Find a maximum flow $f'$ in $N(H', t'; W_1, M)$ from $w_1''$ to $w_2''$, and let $F'$ be the value of $f'$.

(4) Set the tightest lower bound on the weight of $a$ to

$$\frac{max\ \{0, t(b') + M - F\}\ +\ max\ \{0, t'(b'') + M - F'\}}{2}.$$

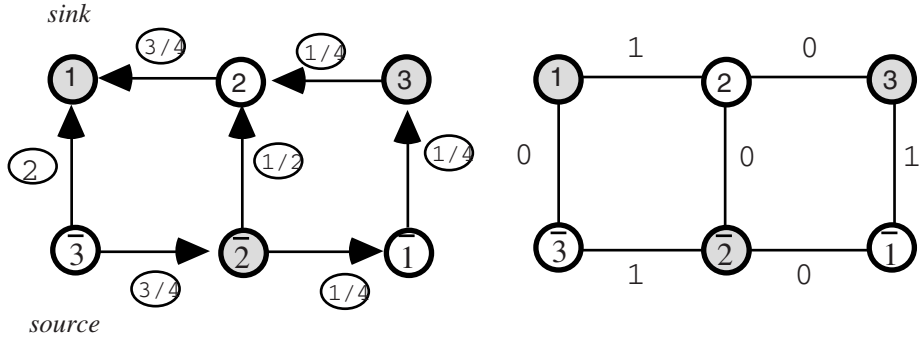
To sum up we have the following

**Theorem 1.** The tightest bounds of an edge of a nonbipartite map can be found with two or four maximum-flow computations depending on whether the arc is a loop or a link.

*Example* 3 (continued). We now apply the procedure above to compute the tightest lower and upper bounds on the weight of the link (1, 3) of the nonbipartite map of Figure 8(*a*). Recall that this arc corresponds to the two arcs ($\overline{3}$, 1) and ($\overline{1}$, 3) of the

associated bipartite map shown in Figure 8(*b*), and that the vertices $\overline{1}$ and $\overline{3}$ are on the side $W_1$ and the vertices 1 and 3 are on the side $W_2$.

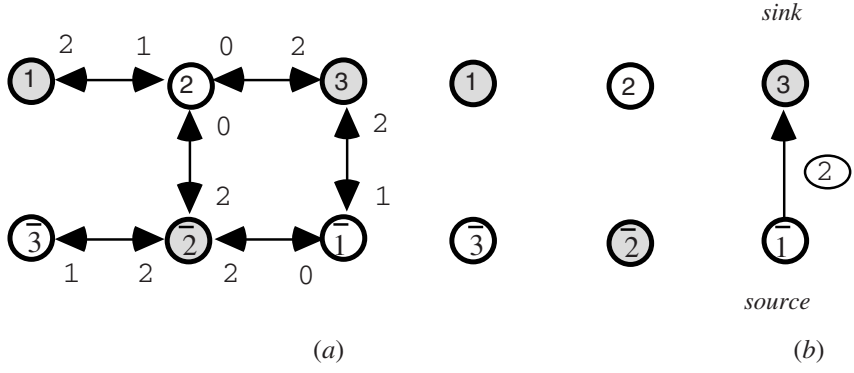
We first apply Algorithm MIN. Figure 9(*a*) shows a maximum flow in the network associated with the bipartite map shown in Figure 8(*b*) from the vertex $\overline{3}$ to the vertex 1, and Figure 9(*b*) shows the corresponding ($\overline{3}$, 1)-minimal variant of the map of Fig. 8(*b*).




(*a*) (*b*)

**Fig. 9.**

So, the tightest lower bound on the weight of the arc ($\overline{3}$, 1) is max $\{0, \frac{3}{4}+2-\frac{11}{4}\}$ = 0. Figure 10(*a*) shows the network associated with the bipartite map of Figure 9(*b*) with the arc ($\overline{3}$, 1) deleted, and Figure 10(*b*) shows a maximum flow in this network from the vertex 3 to the vertex $\overline{1}$.

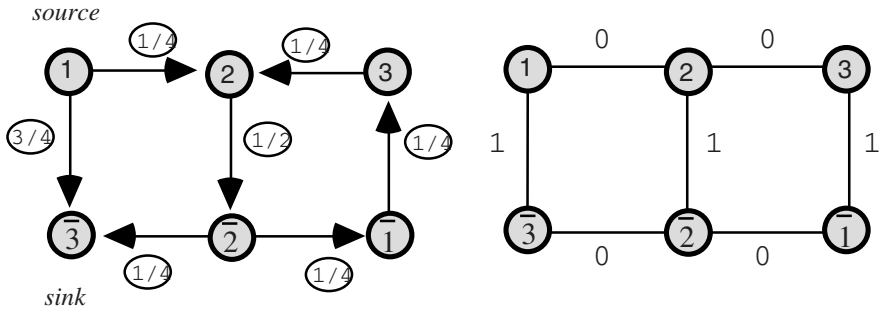



(*a*) (*b*)

**Fig. 10.**

So, the tightest lower bound on the weight of the arc ($\overline{1}$, 3) is max {0, 1+2−2} = 1, and the tightest lower bound on the weight of its corresponding arc (1, 3) is (0+1)/2 = $\frac{1}{2}$.

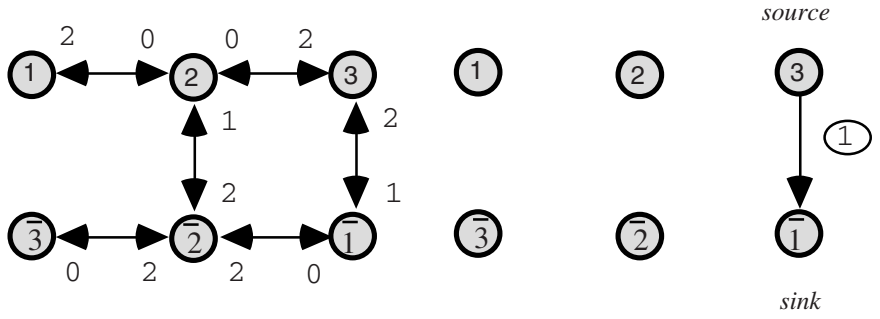
We now apply Algorithm MAX. Figure 11(*a*) shows a maximum flow in the network associated with the bipartite map shown in Figure 8(*b*) from the vertex 1 to the vertex $\overline{3}$, and Figure 11(*b*) shows the corresponding ($\overline{3}$, 1)-maximal variant of the map of Fig. 8(*b*).




(*a*) (*b*)

**Fig. 11.**

So, the tightest upper bound on the weight of the arc ($\overline{3}$, 1) is equal to 1. Figure 12(*a*) shows the network associated with the bipartite map of Figure 11(*b*) with the arc ($\overline{3}$, 1) deleted, and Figure 12(*b*) shows a maximum flow in this network from the vertex 3 to the vertex $\overline{1}$.




(*a*) (*b*)

**Fig. 12.**

So, the tightest upper bound on the weight of the arc ($\overline{1}$, 3) is 1, and the tightest upper bound on the weight of the arc (1, 3) is equal to (1+1)/2 = 1. ■

## 5 A Special Case

In this section, we consider the special case of a complete graph with the addition of one loop for each vertex. We now prove that the tightest bounds on the weight of an edge can be computed with a number of arithmetic operations and comparisons proportional to the number of vertices.

Let $(G, s)$ be a map where $G = (V, A)$ is a complete graph with the addition of one loop for each vertex. Let $q(v) = \sum_{a \in A(v)} s(a)$ for each $v$ in $V$. The constraint system (1) always admits the solution $x$ (see Figure 13) with

$$x(u, v) = \begin{cases} q(v) & \text{if } u = v \\ 0 & \text{else} \end{cases}$$

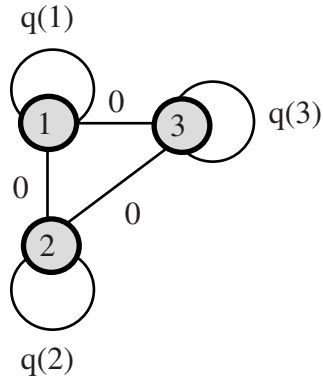



**Fig. 13.**

Moreover, given a link $(u^*, v^*)$ of $G$ with $q(u^*) \leq q(v^*)$, then the constraint system (1) always admits the solution $x$ (see Figure 14) with

$$x(u, v) = \begin{cases} q(u^*) & \text{if } (u, v) = (u^*, v^*) \\ 0 & \text{else} \end{cases}$$

for each link $(u, v)$, and

$$x(u, u) = \begin{cases} 0 & \text{if } u = u^* \\ q(v^*) - q(u^*) & \text{if } u = v^* \\ q(u) & \text{else} \end{cases}$$

for each loop $(u, u)$.

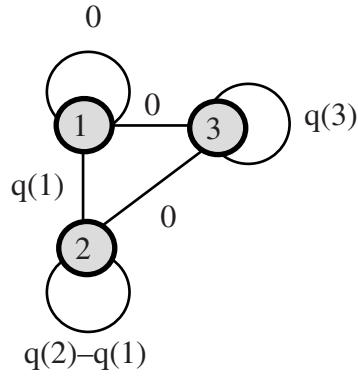



**Fig. 14**.

It follows that the tightest upper bound on the weight of loop $(u, u)$ is equal to $q(u)$, and the tightest lower and upper bounds on the weight of link $(u, v)$ are 0 and $min\ \{q(u), q(v)\}$, respectively. What remains is a formula for the tightest lower bound on the weight of loop $(u, u)$. We now prove that it is given by $max\ \{0, 2q(u) - N\}$, where $N = \sum_{v \in V} q(v)$. Let $(H, t)$ be a bipartite map associated with $(G, s)$ so that the loop $(u, u)$ corresponds to the edge $(u, \overline{u})$ of $H$. For each vertex $w$ of $H$, let $r(w) = q(v)$ if $v$ is the vertex of $G$ corresponding to $w$. By Proposition 3, the tightest lower bound on the weight of $(u, \overline{u})$ is

$$\max\ \{0, r(u) + r(\overline{u}) - \sum_{w \in W_1} r(w)\}\ .$$

But $r(u) = r(\overline{u}) = q(u)$ and $\sum_{w \in W_1} r(w) = \sum_{v \in V} q(v) = N$ so that the statement follows from part $(i)$ of Lemma 2. To sum up, we have the following result.

**Theorem 2**. Let $(G, s)$ be a map where $G$ is a complete graph with the addition of one loop for each vertex. Let $q(v) = \sum_{a \in A(v)} s(a)$ for each $v$ in $V$, and let $N = \sum_{v \in V} q(v)$. Then,

($i$) the tightest lower and upper bounds on the weight of loop $(u, u)$ are respectively $max\ \{0, 2q(u) - N\}$ and $q(u)$ ;

($ii$) the tightest lower and upper bounds on the weight of link $(u, v)$ are respectively 0 and $min\ \{q(u), q(v)\}$.

## 6 Future Research

We considered graphs with arcs weighted by nonnegative reals. The case of integral weights is open; however, since the incidence matrix of a bipartite graph is totally unimodular, the integrality constraint can be relaxed and the results recalled in Section 2 still hold. Another direction for future research should cover the general case of weighted hypergraphs. It is worth mentioning that, for hyperarc weights from {0, 1}, deciding if there exists a hyperarc whose weight is uniquely determined is a co*NP*-complete problem [8].

## References

1. Adam, N.R., Wortmann, J.C.: Security control methods for statistical databases: a comparative study. ACM Computing Surveys **21** (1989) 515–556.
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows. Prentice Hall, Englewood Cliffs, 1993.
3. Chin, F.Y., Ozsoyoglu, G.: Auditing and inference control in statistical databases. IEEE Trans. on Software Engineering **8** (1982) 574–582.
4. Conforti, M., Rao, M.R.: Some new matroids on graphs: cut sets and the max cut problem. Mathematics of Operations Research **12** (1987) 193–204.
5. Cox, L.H.: Suppression methodology and statistical disclosure control. J. American Statistical Association **75** (1980) 377–385.
6. Cox, L.H., Zayatz, L.V.: An agenda for research on statistical disclosure limitation. J. Official Statistics **11** (1995) 205–220.
7. Gusfield, D.: A graph-theoretic approach to statistical data security. SIAM J. Computing **17** (1988) 552–571.
8. Kleinberg, J.M., Papadimitriou, C.H., Raghavan, P.: Auditing Boolean attributes. Proc. XIX ACM Symp. on "Principles of Database Systems" (2000) 86–91.
9. Malvestuto, F.M., Mezzini, M.: A linear algorithm for finding the invariant edges of an edge-weighted graph. SIAM J. on Computing **31** (2002) 1438–1455.
10. Malvestuto, F.M., Mezzini, M.: On the hardness of protecting sensitive information in a statistical database. Proc. World Multiconference on "Systemics, Cybernetics and Informatics", vol. XIV (2001) 504–509.
11. Malvestuto, F.M., Moscarini, M.: Query evaluability in statistical databases. IEEE Transactions on Knowledge and Data Engineering **2** (1990) 425–430.
12. Malvestuto, F.M., Moscarini, M.: An audit expert for large statistical databases. In Statistical Data Protection. EUROSTAT (1999) 29–43.
13. Willenborg, L., de Waal, T.: Statistical Disclosure Control in Practice. Lecture Notes in Statistics, Vol. 111. Springer-Verlag, New York (1996).
14. Willenborg, L., de Waal, T.: Elements of Statistical Disclosure. Lecture Notes in Statistics, Vol. 155. Springer-Verlag, New York (2000).

# CRB-Tree: An Efficient Indexing Scheme for Range-Aggregate Queries*

Sathish Govindarajan, Pankaj K. Agarwal, and Lars Arge

Department of Computer Science, Duke University, Durham, NC 27708
{gsat, pankaj, large}@cs.duke.edu

**Abstract.** We propose a new indexing scheme, called the CRB-tree, for efficiently answering range-aggregate queries. The range-aggregate problem is defined as follows: Given a set of weighted points in $\mathbb{R}^d$, compute the aggregate of weights of points that lie inside a $d$-dimensional query rectangle. In this paper we focus on range-COUNT, SUM, AVG aggregates. First, we develop an indexing scheme for answering two-dimensional range-COUNT queries that uses $O(N/B)$ disk blocks and answers a query in $O(\log_B N)$ I/Os, where $N$ is the number of input points and $B$ is the disk block size. This is the first optimal index structure for the 2D range-COUNT problem. The index can be extended to obtain a near-linear-size structure for answering range-SUM queries using $O(\log_B N)$ I/Os. We also obtain similar bounds for rectangle-intersection aggregate queries, in which the input is a set of weighted rectangles and a query asks to compute the aggregate of the weights of those input rectangles that overlap with the query rectangle. This result immediately improves a recent result on temporal-aggregate queries. Our indexing scheme can be dynamized and extended to higher dimensions. Finally, we demonstrate the practical efficiency of our index by comparing its performance against kdB-tree. For a dataset of around 100 million points, the CRB-tree query time is 8–10 times faster than the kdB-tree query time. Furthermore, unlike other indexing schemes, the query performance of CRB-tree is oblivious to the distribution of the input points and placement, shape and size of the query rectangle.

## 1 Introduction

In order to be successful, any data model in a large database requires efficient external memory (secondary storage) support for its language features. Range searching and its variants are problems that often need to be solved efficiently [16]. In on-line analytical processing (OLAP), spatial databases such as geographic information systems (GIS), and several other applications, range-aggregate queries (e.g., range-COUNT, range-SUM, etc) play an extremely important role, and a large number of algorithms and indexing structures have been proposed to answer such queries; see e.g. [10] and references therein. With the rapid increase in the use of data warehouses to collect historical information,

* The first two authors are supported by Army Research Office MURI grant DAAH04-96-1-0013, by NSF grants ITR–333–1050, EIA–9870724, EIA–997287, and CCR–9732787, and by a grant from the U.S.-Israeli Binational Science Foundation. The third author is supported by the National Science Foundation through ESS grant EIA–9870734, RI grant EIA-9972879, CAREER grant CCR-9984099, and ITR grant EIA-0112849.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 143–157, 2003.

temporal aggregate queries have also received much attention in the last few years [21, 22]. In general, answering temporal (or bi-temporal) aggregate queries is harder than traditional aggregate queries because each data point is associated with a time interval during which its attribute values are valid and a query calls for computing an aggregate over only those points that are valid during the query time interval.

In this paper we present an optimal indexing scheme for answering 2D range-`COUNT` queries. Our index can be extended to efficiently answer multidimensional range-aggregate queries. We can also efficiently answer several important temporal aggregate queries using our index.

*Model.* Let $P$ be a set of $N$ points in $\mathbb{R}^d$, and let $w : P \to \mathbb{Z}$ be a *weight* function. We wish to build an index on $P$ so that we can efficiently answer *range-aggregate* queries, i.e., compute the aggregate of weights of points in $P$ that lie inside a given $d$-dimensional query rectangle $R$. In this paper, we focus on *range-aggregate* queries, such as range-`COUNT`, `SUM`, `AVG`. These three examples call for computing $|P \cap R|$, $\sum_{p \in P \cap R} w(p)$, and $\sum_{p \in P \cap R} w(p)/|P \cap R|$, respectively; see Figure 1(i). We also study a more general problem in which $P$ is a set of rectangles in $\mathbb{R}^d$. A query is also a $d$-dimensional rectangle and the goal is to compute an aggregation over the set of input rectangles that overlap with the query rectangle; see Figure 1(ii). We refer to this problem as the *rectangle-intersection-aggregate* problem. This problem arises naturally when a massive set of points is replaced by a set of rectangles (the weight of a rectangle being the average weight of the data points inside that rectangle) or when an input is a set of complex objects and each object is replaced by its minimum bounding box. In the *temporal range-aggregate* problem, studied in [21,22], we are given a set $P$ of (time) intervals in $\mathbb{R}^d$ and the goal is to compute an aggregation over the set of intervals that intersect a query rectangle; see Figure 1(iii). The range-aggregate problem is a special case of the temporal range-aggregate problem, which in turn is a special case of the rectangle-intersection-aggregate problem.

As our main interest is minimizing the number of disk blocks used by the index and the number of disk accesses used to answer a query, we will consider the problem in the standard external memory model [2]. This model assumes that each disk access transmits a contiguous block of $B$ units (words) of data in a single *input/output operation* (or *I/O*). The efficiency of an index is measured in terms of the amount of disk space it uses (measured in units of disk blocks) and the number of I/Os required to answer a query, to

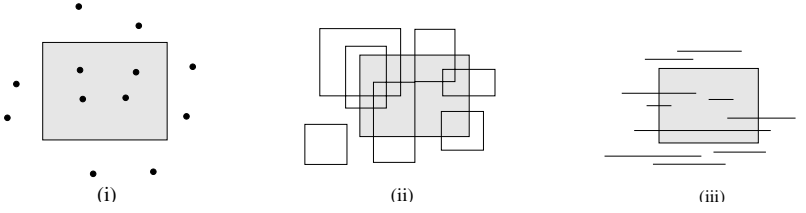



**Fig. 1.** (i) Range-aggregate query, (ii) Rectangle-intersection-aggregate query, (iii) Temporal range-aggregate query.

bulkload (construct) the index, and to update the index. The minimum number of disk blocks we need to store $N$ points is $n = \lceil N/B \rceil$, which we refer to as "linear" size. We also assume that the size of internal memory $M$ is at least $B^2$ and that the integer $N$ can be stored in a single word, i.e., that a word can store $\lceil \log N \rceil$ bits and each bit can be accessed individually. The latter assumption is made in any reasonable computational model and is important for our space- and query-efficient index.

*Related work.* There has been a lot of work in the spatial database community on indexing a set of points for answering range queries. Indexing schemes such as numerous variants of $kdB$-trees and R-trees, external priority search trees, etc. have been proposed; see [3, 16] for recent surveys. A $kdB$-tree uses $O(n)$ disk blocks and can be used to answer a range-aggregate query in $O(\sqrt{n})$ I/Os. No worst-case bound on the query performance of the R-tree is known. Recently, Tao et al. [18] proposed a aP-tree that uses $O(n \log_B n)$ disk blocks and answers a range-aggregate query using $O(\log_B n)$ I/Os.

As temporal aggregation is being included in most of the temporal languages, there has been a flurry of activity on answering temporal aggregate queries. After several early results, including [14,13], Yang and Widom [21] proposed an indexing scheme called the SB-tree that stores a set of $N$ "time" intervals in $\mathbb{R}^2$ using $O(n \log_B n)$ disk blocks so that for a time interval $\Delta$, the aggregate over all keys that are valid at some time in the interval $\Delta$ can be computed using $O(\log_B n)$ I/Os. Recently, Zhang *et al.* [22] proposed a *multiversion* SB-tree (MVSB-tree) that uses $O(n \log_B n)$ disk blocks in the worst case and that can answer a temporal range-aggregate query in $O(\log_B n)$ I/Os. Their index can also answer range-aggregate queries such as range-`COUNT` and range-`SUM` in $O(\log_B n)$ I/Os. The SB-tree and MVSB-tree also support updates in $O(\log_B n)$ I/Os.

There is also a vast literature on answering range-aggregate queries in OLAP systems, in which the data is typically modeled as a multidimensional cube and queries typically involve aggregation across various cube dimensions [6,11,12,15]. Although the data cube is a good data model for OLAP systems, it only works well when the data is dense in the sense that the data in a $d$-dimensional data cube with a total of $N$ cells has more than $N^{1/d}$ distinct keys (on average) in each dimension. In many applications such as spatial databases, data tends to be sparse and data cube model does not work well.

Several range-searching data structures in the internal memory model have been developed in computational geometry; see [1] for a survey. The best-known data structure in $\mathbb{R}^2$ is the *range tree*, which uses $O(N \log_2 N)$ space and can answer a range-aggregate query such as range-`SUM` in $O(\log_2 N)$ time [1]. Chazelle [7] developed the *compressed range-tree* data structure that uses $O(N)$ space under the so-called bit model (in which a word can store $\log_2 N$ bits and each bit can be manipulated individually). This structure can be used to answer a range-`COUNT` query in $O(\log_2 N)$ time. Both of these structures use subtraction to answer queries. The compressed range trees can be extended to other range-aggregate queries by paying a polylogarithmic overhead in the query time. If we do not allow subtraction, then the lower-bound results by Chazelle [8] in the semigroup model (see e.g. [1,8] for a precise definition of this model) suggest that a structure that answers a 2D range-`COUNT` query in $O(\log_2 N)$ time has to use super-linear storage.

*Our results.* Our main result, described in Section 2, is a new indexing scheme, called the *Compressed Range $B$-tree* (or *CRB-tree*), for answering two-dimensional range-`COUNT` queries. This structure is an external version of the compressed range-tree [7].

It uses $O(n)$ disk blocks, answers a query in $O(\log_B n)$ I/Os, and can be bulk-loaded using $O(n \log_B n)$ I/Os. This is the first optimal indexing scheme for the 2D range-COUNT problem in the I/O model. Using a partial-rebuilding scheme [5], a point can be inserted/deleted in $O(\log_B^2 n)$ I/Os. Section 3 presents several extensions of our basic structure. We first adapt the structure for answering 2D range-SUM queries in $O(\log_B n)$ I/Os using $O(n \log_B((W \log_2 W)/N))$ disk blocks, where $W = \sum_{p \in S} w(p)$ is the total weight of the input points. It can also answer other range-aggregate queries such as range-MAX,MIN in $O(\log_B^2 n)$ I/Os. Next we extend our index to higher dimensions. In $\mathbb{R}^d$, the structure uses $O(n \log^{d-2} n)$ disk blocks, answers a range-COUNT query in $O(\log_B^{d-1} n)$ I/Os, and can be bulk loaded in $O(n \log_B^{d-1} n)$ I/Os. Similar bounds can be derived for other range-aggregate queries. Using a result by Edelsbrunner and Overmars [9], our index can also be used to answer rectangle-intersection-COUNT(resp. SUM) queries without affecting the asymptotic bound. Since temporal range-aggregate queries are a special case of rectangle-intersection aggregate queries, our structure improves upon the recent results by Zhang et al. [22] when the total weights of the points is $O(N)$. More precisely, we can answer a temporal range-aggregate query in $O(\log_B n)$ I/Os using an $O(n \log_B(\log_2 N))$-size index.

We have implemented the two-dimensional CRB-tree and in Section 4 we report the results of an extensive experimental evaluation of its efficiency. Since we are mainly interested in linear-size indexes, we compare the performance of the CRB-tree with that of the $kdB$-tree. We have evaluated the performance of these index structures using synthetic and TIGER/Line data. Our first set of experiments study the query and bulk loading performance on datasets of size ranging from 20 to 140 million points. Our experiments show that the query performance of CRB-trees is significantly better than that of kdB-trees. For a data set with around 100 million points, the CRB-tree query time is 8–10 times faster than the kdB-tree query time. Our second set of experiments compare the query performance by varying the size and shape of the query rectangle. The query performance of CRB-trees is independent of the shape and size of the query rectangle while the query time of kdB-tree increases rapidly with the size of the query rectangle.

## 2 CRB-Tree

In this section, we describe the *Compressed Range B-tree* (CRB-tree), an indexing scheme for answering two-dimensional range-COUNT queries. The structure is an external version of an internal memory structure due to Chazelle [7]. We first describe the CRB-tree and how to bulkload it, and then present the query procedure.

*CRB-tree.* Let $P$ denote the set of $N$ points in the plane. A CRB-tree consists of two structures: a $B^+$-tree $T$ constructed on the $x$-coordinates of $P$, with each internal node $v$ storing a secondary structure $\Sigma_v$, and a normal $B^+$-tree $\Psi$ constructed on the $y$-coordinates of $P$ (Figure 2 shows an example of a CRB-tree $T$ on $N = 16$ points.)

Let $P_v = \{p_1, p_2, \dots\}$ denote the sequence of points contained in the subtree of $T$ rooted at $v$, sorted in a non-decreasing order of their $y$-coordinates. Set $N_v = |P_v|$ and $n_v = N_v/B$. For any $y$-value $y_0$, the secondary structure $\Sigma_v$ will be used to count the number of points of $P_v$ that "belong" to a given child of $v$ and whose $y$-coordinates are at

most $y_0$. Intuitively, for each child $v_i$ of $v$, $\Sigma_v$ should maintain an array whose $j$th entry, for $j \leq N_v$, stores the count, i.e., how many points among the first $j$ points of $P_v$ belong to $P_{v_i}$. Using these "prefix sums", we can determine in one I/O the number of points of $P_v$ that belong to $P_{v_i}$ and whose $y$-coordinates are less than $y_0$. However, storing these prefix sums requires $O(n_v)$ disk blocks, which would lead to an overall space of $O(n \log_B n)$ blocks. We therefore store the "prefix sum" array compactly, using only $O(n_v/(\log_B n))$ blocks. Roughly speaking, we partition $P_v$ into consecutive *"chunks"* and compute the prefix-sums only at the *"chunk"* level, i.e., for the $l$th chunk and for each child $v_i$ of $v$, we store the number of points in the first $l$ *chunks* that belong to $P_{v_i}$. Let the predecessor of $y_0$ in $P_v$ belong to the $k$th *chunk* of $P_v$. The desired count is the sum of the "prefix sum" till the $(k-1)$th *chunk* and the number of points within the $k$th *chunk* that belong to $P_{v_i}$ and whose $y$-coordinates are less than $y_0$. We also preprocess each *chunk* separately so that we can compute the latter count using $O(1)$ I/Os.

More precisely, $\Sigma_v$ consists of two arrays — a *child index* array $CI_v$ and a *prefix count* array $PC_v$. Let $v_0, v_1, \ldots, v_{B-1}$ be the children of $v$. We regard the child index array $CI_v$ as a one-dimensional array with a $\log_2 B$ bit entry for each of the $N_v$ points in $P_v$. $CI_v[i]$ simply stores the index $b_i$ of the subtree of $v$ storing the $i$th point $p_i$ of $P_v$, i.e., $p_i$ is stored at a leaf in the subtree rooted at $v_{b_i}$. (For example, in Figure 2 (iv), $CI_v[3] = 1$ since the third point in $P_v$, $(7, 3)$, belongs to the subtree of $v_1$). Since $b_i < B$, the $\log_2 B$ bits of the $i$th entry are enough to represent $b_i$. $CI_v$ requires $N_v \log_2 B$ bits and thus can be stored using $\frac{N_v}{B} \log_2 B/(\log_2 N) = n_v/\log_B N$ blocks. Next, let $\mu = B \log_B N$ and $r = N_v/\mu$. (Note that $\mu$ entries of $CI_v$ fit in one block). We partition the $CI_v$ array into *"chunks"* of length $\mu$ and store the prefix sum at the *chunk* level in $PC_v$, the prefix count array. We regard $PC_v$ as a two-dimensional array with $r$ rows and $B$ columns, with each entry storing one word (that is, $\log_2 N$ bits). For $1 \leq i \leq r$ and $0 \leq j < B$, $PC_v[i, j]$ stores the number of points among the first $i\mu$ points of $P_v$ that belong to $P_{v_j}$, $PC_v[i, j] = |\{p_1, \ldots, p_{i\mu}\} \cap P_{v_j}|$. (For example, in Figure 2(iv), $\mu = 8$ and $PC[1, 0] = 2$ since among the first 8 points of $P_v$, only 2 points belong to $v_0$.) $PC_v$ requires $rB = N_v/\log_B N$ words. Hence $\Sigma_v$ can be stored using $O(n_v/\log_B n)$ disk blocks. Since the height of $T$ is $O(\log_B n)$ and a point of $P$ is stored only at one leaf of $T$, we have that $\sum_{v \in T} n_v = O(n \log_B n)$. The total space used by $T$ is thus $\sum_{v \in T} n_v/\log_B n = O(n)$ disk blocks.

*Bulk loading a CRB-tree.* A CRB-tree can be bulk-loaded efficiently bottom-up, level by level. We construct the leaves of $T$ using $O(n \log_{M/B} n)$ I/Os by sorting the points in $P$ in non-decreasing order of their $x$-coordinates [2]. We then sort the points stored at each leaf $v$ in non-decreasing order of their $y$-coordinates to get $P_v$. Below we describe how we construct all nodes and secondary structures at level $i$ of $T$, given that we have already constructed the nodes at level $i + 1$.

We construct a level $i$ node $v$ and its secondary secondary structure $\Sigma_v$ as follows: We first, compute $P_v$ (sorted by the $y$-coordinates) by merging $P_{v_0}, P_{v_1}, \ldots, P_{v_u}$, where $v_0, \ldots, v_u$ are the children of $v$. Since $M > B^2$, the internal memory can hold a block from each $P_{v_i}$ at the same time. This means that we can compute $P_v$ using a single scan through the $P_{v_i}$'s, i.e., in $O(n_v)$ I/Os. Recall that $CI_v[i]$ contains the index of the child containing $p_i$. In order to construct $CI_v[i]$ we record the index of the child of $v$ from which each point of $P_v$ came from. This way we can construct $CI_v$ using a single scan

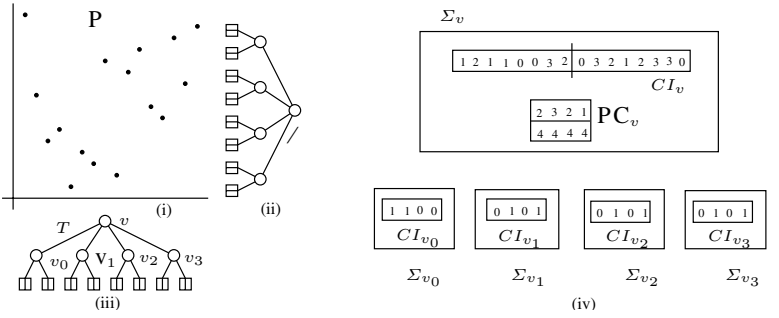



**Fig. 2.** CRB-tree for a set of $N = 16$ points.
(i) $P = \{(5,1),(9,2),(7,3),(6,4),(3,5),(4,6),(13,7),(12,8),(2,9),(15,10),(10,11),(8,12),(11,13),(14,14),(16,15),(1,16)\}$ (ii) B-tree $\Psi$ on the $y$-coordinates of $P$;B=4 and each leaf stores 2 points of $P$. (iii) B-tree $T$ on the $x$-coordinates of $P$;B=4 and each leaf stores 2 points of $P$. $P_v = P$, $P_{v_0} = \{(3,5),(4,6),(2,9),(1,16)\}$, $P_{v_1} = \{(5,1),(7,3),(6,4),(8,12)\}$, $P_{v_2} = \{(9,2),(12,8),(10,11),(11,13)\}$ , $P_{v_3} = \{(13,7),(15,10),(14,14),(16,15)\}$ (iv) Secondary structure stored at each internal node of $T$. $\mu = 8$; since each child of the root stores $4(< 8)$ points, no entries are stored in the PC array at those nodes. $PC_v[1,j]$ (resp. $PC_v[2,j]$) is $k$ if $k$ points among the first 8 (resp. 16) points of $P$ are stored at the child $v_j$ of the root.

through $P_v$. Since $PC_v[i,j]$ contains the number of points among the first $i\mu$ points in $P_v$ that belong to $P_{v_j}$, we can also construct $PC_v$ in a single scan through $CI_v$ as follows: We compute $PC_v[i,j]$ while scanning the points $p_{(i-1)\mu+1},\ldots,p_{i\mu}$. Since $0 \le j \le B-1$, we maintain $PC_v[i,0], PC_v[i,1],\ldots,PC_v[i,B-1]$ in internal memory. If $i = 1$, we initialize $PC_v[i,j] = 0$, otherwise we initially set $PC_v[i,j] = PC_v[i-1,j]$. If we are currently scanning $p_k$ and $CI_v[k] = j$ we increment $PC_v[i,j]$. After we have scanned $p_{i\mu}$, we store $PC_v[i,0], PC_v[i,1],\ldots,PC_v[i,B-1]$ to disk.

Since the number of I/Os used to construct a level $i$ node $v$ and its secondary structure $\Sigma_v$ is $n_v$, the total number of I/Os used to build level $i$ of $T$ is $O(n)$. Thus we can construct the CRB-Tree using $O(n \log_B n)$ I/Os in total.

*Answering queries.* Let $Q = [x_1,x_2]\times[y_1,y_2]$ be a query rectangle. We compute $|P \cap Q|$ by traversing $T$ in a top-down manner, visiting $O(\log_B n)$ nodes. The query procedure traverses $T$ along two paths , namely the paths from the root to the leaves $w$ and $z$ containing $x_1$ and $x_2$, respectively. We use the secondary structures along the two paths to answer the query. To explain how, imagine associating an $x$-interval $I_v$ with each node $v$ of $T$. At the root $u$ of $T$, $I_u$ is the entire $x$-axis and the interior of the node $v$ partition $I_v$ into $B$ $x$-intervals. We associate the $i$th interval with the $i$th child of $v$. Consider the topmost node $v$ such that $I_v$ contains $[x_1,x_2]$ but none of the $x$-intervals $I_{v_j}$ associated with a child of $v$ contains $[x_1,x_2]$; $v$ is the nearest common ancestor of $w$ and $z$. Let $v_\lambda$ and $v_\rho$ be the two children of $v$ such that $x_1$ and $x_2$ lie in the intervals $I_\lambda$ and $I_\rho$ respectively. (Figure 3 (i) shows the intervals for the children of the root node of $T$. For the given $Q$, $\lambda = 0, \rho = 3$ at the root.) Obviously $|P \cap Q| = \sum_{\lambda \le j \le \rho} |P_{v_j} \cap Q|$. To answer the query we compute the count $C = \sum_{\lambda < j < \rho} |P_{v_j} \cap Q|$ at $v$ using the secondary structure $\Sigma_v$ and recursively visit the children $v_\lambda$ and $v_\rho$ to compute $|P_{v_\lambda} \cap Q|$ and $|P_{v_\rho} \cap Q|$. (As shown in Figure 3(i), we compute the counts $|P_{v_1} \cap Q|$ and $|P_{v_2} \cap Q|$ at $v$, and

recursively compute $|P_{v_0} \cap Q|$ and $|P_{v_3} \cap Q|$.) Below we show how we can compute the count $|P_{v_i} \cap Q|$ at one node in $O(1)$ I/Os. When we reach a leaf $w$, we compute $|P_w \cap Q|$ in $O(1)$ I/Os by scanning all the (at most $B$) points in $P_v$. Since we visit $O(\log_B n)$ nodes we need $O(\log_B n)$ I/Os to answer a query.

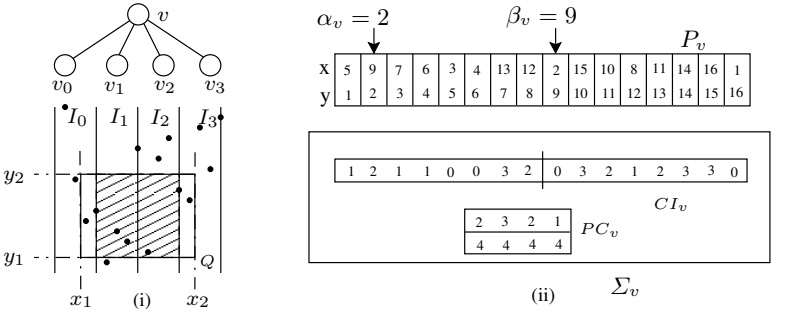



**Fig. 3.** (i) The query rectangle $Q = [2.5, 13.5] \times [1.5, 9.5]$ along with the input points $P$ drawn in Figure 2. $I_j$ is the interval associated with the child $v_j$ of the root. $\lambda = 0, \rho = 3$. $|P_{v_0} \cap Q|$, $|P_{v_3} \cap Q|$ are computed recursively and $|P_{v_1} \cap Q|$, $|P_{v_2} \cap Q|$ are computed at the root. (ii) $P_v$ and the secondary structure $\Sigma_v$ at the root.

In order to compute the above count, we maintain two variables $\alpha_v$ and $\beta_v$ when visiting a node $v$ during the traversal of the tree: $\alpha_v$ is the rank of the first point in $P_v$ whose $y$-coordinate is at least $y_1$, and $\beta_v$ is the rank of the last point in $P_v$ whose $y$-coordinate is not larger than $y_2$. For $0 \leq j \leq B-1$ and $1 \leq r \leq N_v$, let $\varphi(j, r)$ denote the number of points in $P_v$ of rank at most $r$ that belong to $P_{v_j}$, i.e., $\varphi(j, r)$ is the number of entries among the first $r$ entries of $CI_v$ that store the index $j$. For $\lambda < j < \rho$, since the $x$-coordinates of $P_{v_j}$ lie in the range $[x_1, x_2]$, $|P_{v_j} \cap Q| = \varphi(j, \beta_v) - \varphi(j, \alpha_v)$. (For example, from the $P_v$ array in Figure 3 (ii), $\varphi(1, 9) = 3$, since three points among the first nine points of $P_v$ belong to $P_{v_1}$. Similarly, $\varphi(1, 2) = 1$. Thus we have $|P_{v_1} \cap Q| = 3 - 1 = 2$. From Figure 3 (i), we see that $|P_{v_1} \cap Q| = 2$ since there are two points of $P_v$ in the shaded region of child $v_1$.) It thus suffices to describe how to maintain variables $\alpha_v$ and $\beta_v$ and how to compute $\varphi(0, r), \ldots, \varphi(B-1, r)$.

We can compute $\alpha_v$ and $\beta_v$, at the root of $T$ in $O(\log_B n)$ I/Os by searching for $y_1$ and $y_2$ in the B-tree $\Psi$. $\alpha_v$ and $\beta_v$ are the ranks of the points at which the search terminates. Assuming we have computed $\alpha_v$ and $\beta_v$ at a node $v$ of $T$, we can compute $\alpha_{v_j}$ and $\beta_{v_j}$ for all children $v_j$ of $v$ as follows. Since $\alpha_{v_j}$ is the rank of the first point in $P_v$ that belongs to $P_{v_j}$ and whose $y$-coordinate is at least $y_1$, $\alpha_{v_j} = \varphi(j, \alpha_v)$ and $\beta_{v_j} = \varphi(j, \beta_v)$. Thus the problem of maintaining $\alpha_v, \beta_v$ reduces to computing $\varphi(j, \alpha_v)$ and $\varphi(j, \beta_v)$ for all $0 \leq j \leq B-1$.

All that remains is to describe the procedure for computing $\varphi(0, r), \ldots, \varphi(B-1, r)$ for a given $r$ in $O(1)$ I/Os. Suppose $r = \mu a + c$ for $a \geq 0$ and $0 \leq c \leq \mu$. Then

$$\begin{aligned} \varphi(j, r) &= |\{k \mid k \leq r \text{ and } CI_v[k] = j\}| \\ &= PC_v[a, j] + |\{k \mid \mu a < k \leq r \text{ and } CI_v[k] = j\}|. \end{aligned} \quad (1)$$

Thus our procedure simply reads the two disk blocks storing $PC_v[a,1],\ldots,PC_v[a,B]$ and $CI_v[\mu a+1],\ldots,CI_v[r]$, respectively, and then computes $\varphi(0,r),\ldots,\varphi(B-1,r)$ using (1). For example, let us calculate $\varphi(2,11)$. From the *PC* and *CI* array in Figure 3 (ii), $\varphi(2,11) = PC[1,2]+1 = 3$. From the $P_v$ array in Figure 3 (i), there are three points among the first eleven points that belong to $P_{v_2}$.

**Theorem 1.** *A set of $N$ points in the plane can be stored in a linear-size index structure using $O(n\log_B n)$ I/Os so that a range-`COUNT` query can be answered in $O(\log_B n)$ I/Os.*

We can make the CRB-tree structure dynamic by slightly modifying the external-memory logarithmic method of Arge and Vahrenhold [5]. Omitting all the details from this abstract, we obtain the following.

**Theorem 2.** *A set of $N$ points in the plane can be stored in a linear-size index structure so that a range-`COUNT` query can be answered in $O(\log_B^2 n)$ I/Os and a point can be inserted or deleted in $O(\log_B^2 n)$ amortized I/Os.*

## 3 Extensions

In this section we discuss various extensions of CRB-trees. We present the main ideas in these extensions and omit the details.

*Range-`SUM` queries.* We first discuss how to extend CRB-trees to answer range-`SUM` queries in the plane. Let $P$ be a set of $N$ points in $\mathbb{R}^2$, and let $w : P \to \mathbb{Z}$ be the weight function. If the weight of each point can be stored using $O(1)$ bits, then we can easily modify the CRB-tree by storing the weights in an additional array similar to $CI$ and storing the prefix sum of the weights in another array similar to $PC$. So we focus on the case in which the weights of the points vary considerably. Set $W = \sum_{p\in P} w(p)$ and $\omega = \log_B(\log_2 W)+\log_B(W/N)$. We extend the CRB-tree by storing four additional arrays $W_v, CW_v, L_v$, and $CL_v$ in the secondary structure $\Sigma_v$ of each internal node $v$ of the B-tree. Let $w_i = w(p_i)$ be the $i$th point $p_i \in P_v$. Set $s_i = \max\{\log_2\log_2 W, \log w_i\}$; $s_i \le \log_2 W$. We store $w_i$ in the array $W_v$, using $s_i$ bits, as a continuous sequence of bits. $W_v$ requires at most $\mu_v = \sum_i s_i \le N_v \log_2\log_2 W + N_v\log_2(W/N)$ bits and thus $\mu_v = O(n\omega/\log_B n)$ disk blocks. $W_v$ plays the role of $CI_v$. Since $W_v$ is stored as a packed array, we need two additional arrays $L_v$ and $CL_v$ to determine the index in $W_v$ that stores the leftmost bit of $w_i$, for any given $i \le N_v$. $L_v$ is an array of length $N_v$, each entry composed of $\log_2\log_2 W$ bits. $L_v[i]$ stores the value of $s_i$. Since $s_i \le \log_2 W$, it needs at most $\log_2\log_2 W$ bits. The size of $L_v$ is thus $O(\frac{N_v}{B}\log_2(\log_2 W)/(\log_2 N)) = O(n_v\log_B(\log_2 W)/(\log_B n))$ disk blocks. $CL_v$, an array of length $O(n\omega/\log_B n)$ blocks, stores the prefix sum of $L_v$, as $PC_v$ in Section 2, so that the leftmost bit of $w_i$ in $W_v$, for any $i \le N_v$, can be computed using $O(1)$ I/Os. Finally, $CW_v$ stores prefix sum of weights in $W_v$, as $PC_v$ in Section 2, so that for any $i \le N_v$, one can compute the sum of weights of points in $\{p_1,\ldots,p_i\}\cap P_w$ for all children $w$ of $v$. Since the details are similar to Section 2, we omit them from this abstract and conclude the following.

**Theorem 3.** *Let $P$ be a set of $N$ points in $\mathbb{R}^2$, let $w : P \to \mathbb{Z}$ be a weight function, and let $W = \sum_{i=1}^{N} w_i$. We can bulk load an index in $O(n \log_B((W \log_2 W)/N) \log_B n)$ I/Os that uses $O(n \log_B((W \log_2 W)/N))$ disk blocks so that a range-SUM query can be answered using $O(\log_B n)$ I/Os. If the weight of each point requires $O(1)$ bits, then the size and bulk-loading bounds are $O(n)$ and $O(n \log_B n)$, respectively.*

*Indexing in higher dimensions.* The CRB-tree can be extended to answer range-aggregate queries in $\mathbb{R}^d$ by constructing a multi-level tree structure as follows. Again, we focus on range-COUNT queries. Let $P$ be a set of $N$ points in $\mathbb{R}^d$, and set $b = B^{1/(d-1)}$. A $d$-dimensional CRB-tree is a B-tree $T^d$, with fanout $b$, built on the $x_d$-coordinates of $P$. Each internal node $v$ of $T^d$ is associated with a subset $P_v$ of $P$ and stores a secondary structure, which is a $(d-1)$-dimensional CRB-tree $T^{d-1}$ on the projection of $P_v$ onto the hyperplane $x_d = 0$. The recursion stops when we have built the two-dimensional CRB-tree (with fanout $b$) in the $x_1x_2$-plane. As in the 2D case, the secondary structure of each internal node $v$ of $T^2$ stores two arrays $CI_v$ and $PC_v$, though each entry now keeps more information.

Let $T^2$ be a two-dimensional CRB-tree and let $v$ be an internal node of $T^2$. We associate a $(d-1)$ tuple $(w_d, w_{d-1}, \ldots, w_2)$ with $v$ where $w_2 = v$ and $w_i$ is a node of the $i$-dimensional CRB-tree to which $T^2$ is attached. For each point $p \in P_v$, and for $2 \le i \le d$, let $w_{a_i}$ be the child of $w_i$ (in the $i$-dimensional tree) so that $p \in P_{w_{a_i}} \subset P_{w_i}$. We call $(a_d, \ldots, a_2)$ the child-index sequence of $p$. $CI_v$ is a two-dimensional array with $N_v$ rows and $d-1$ columns. The row corresponding to the point $p \in P_v$ stores the child-index sequence of $p$. $CI_v$ requires $N_v(d-1)\log b = O(N_v \log B)$ bits. $PC_v$ stores the prefix sum of $CI_v$, as in Section 2. The total size of $T^d$ is $O(n \log^{d-2} n)$ disk blocks and can be constructed using $O(n \log^{d-1} n)$ I/Os.

Let the query rectangle $Q$ in $\mathbb{R}^d$ be $[\alpha_1, \beta_1] \times \cdots \times [\alpha_d, \beta_d]$. The query procedure in $T_d$ follows two paths to the leaves corresponding to the points $\alpha_d$ and $\beta_d$. For each node $v$ on these paths, the procedure recursively visits the $(d-1)$-dimensional CRB-tree stored at $v$ .When we reach a node $v$ of $T^2$, for all children $w$ of $v$, we count $|P_w \cap [\alpha_1, \beta_1] \times \cdots [\alpha_d, \beta_d]|$. This can be reduced to answering the following query: given a real value $\alpha$ and a $(d-1)$-tuple $\omega = (\omega_d, \ldots, \omega_2)$, count all the points in $P_w$ whose $x_1$-coordinates are at most $\alpha$ and whose child-index sequence is $\omega$. We use arrays $CI_v$ and $PC_v$ to answer the above query efficiently. Omitting all the details, we conclude the following.

**Theorem 4.** *Let $P$ be a set of $N$ points in $\mathbb{R}^d$. We can bulk load an index on $P$ using $O(n \log_B^{d-1} n)$ I/Os that uses $O(n \log_B^{d-2} n)$ disk blocks and can answer a $d$-dimensional range-COUNT query in $O(\log_B^{d-1} n)$ I/Os.*

*Rectangle-intersection aggregate queries.* CRB-trees can be extended to answer rectangle-intersection aggregate queries by using the reduction of Edelsbrunner and Overmars [9] to transform rectangle-intersection COUNT queries in $\mathbb{R}^d$ to range-COUNT queries in $\mathbb{R}^d$. Omitting all the details, we have the following result.

**Theorem 5.** *Let $P$ be a set of $N$ rectangles in $\mathbb{R}^d$. We can bulk load an index on $P$ using $O(n \log_B^{d-1} n)$ I/Os that uses $O(n \log_B^{d-2} n)$ disk blocks and can answer a $d$-dimensional range-intersection-COUNT query in $O(\log_B^{d-1} n)$ I/Os.*

# 4 Experimental Results

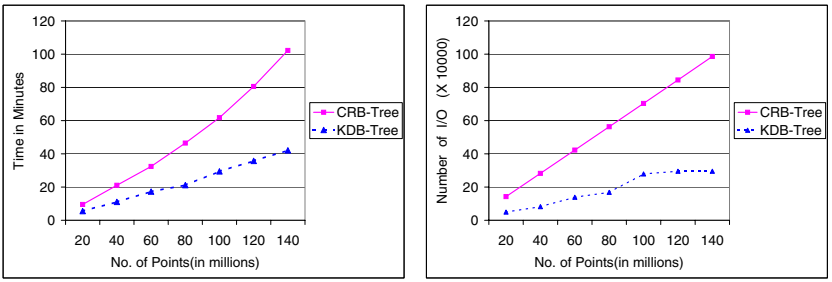
In this section we report the results of an extensive set of experiments with the CRB-tree. The emphasis of our experiments is on the size and query time of the index, and therefore we provide experimental results only for 2D range COUNT queries. Since we are mainly interested in linear space indexes, we chose to compare the performance of the CRB-tree with that of the $kdB$-tree [17], and not e.g. with the MVSB-tree which uses $O(n \log_B n)$ space [22]. In the full version of this paper, we will provide experimental results on SUM queries and as well as on dynamization.

## 4.1 Implementation

We implemented the CRB-tree using the TPIE system developed at Duke. TPIE is designed to facilitate easy and portable implementations of I/O-efficient algorithms and indexing structures, and consists of a set of templated C++ classes and functions. The TPIE system consists of a stream and a block oriented part [20,4]. In the stream oriented part, user programs are fed a continuous stream of elements in an I/O-efficient manner. In the block oriented part, the external memory is viewed as a collection of blocks and primitives for manipulating such blocks are provided. Both the CRB-tree and kdB-tree implementation use both parts of TPIE. The nodes of the B-tree and kdB-tree are implemented using blocks. The stream oriented part is used for efficiently implementing the bulk loading algorithm of both the indexes.

For the CRB-tree, the block size of $8K$ bytes allowed for a fanout of 500 and a maximum leaf size of 681. The precise number of blocks used for the CRB-tree, can be roughly estimated to $4n$; $n$ blocks for each of the base tree $T$, $\Psi$, and the secondary structure arrays $CI$ and $PC$. The arrays $CI$ and $PC$ were also implemented using blocks. We implemented the arrays such that the entries needed to compute the count $C$ at any node of $T$ can be loaded using only four I/Os (loading two blocks of $CI$ and $PC$ array). Thus the query process uses 5 I/Os at each node of $T$ (4 I/Os to access the secondary structure and 1 I/O to access the node). The total number of nodes of $T$ accessed by the query procedure is almost $2 \log_B n - 1$, since the query search corresponds to 2 root-to-leaf paths in $T$. The same is true for the number of nodes of $\Psi$ accessed by the query. Thus the total number of I/Os performed by the query procedure is $5(2 \log_B n - 1) + (2 \log_B n - 1) = 6(2 \log_B n - 1)$.

Since each node $v$ of our $kdB$-tree stores a balanced binary tree of height 8 whose leaves are the children of $v$, the $8K$ block size allowed for a fanout of 255 and a maximum leaf size of 681. The number of blocks used for the $kdB$-tree can be roughly estimated to be $n$. We bulk-load the $kdB$-tree using a top-down approach. At each node of the $kdB$-tree, we store the count of the number of points contained in the subtree of each of its children. The query process traverses the $kdB$-tree, starting from the root. At each node $v$, it checks which regions corresponding to $v$'s children intersects the query region. The query process recurses on those childrens whose region is intersected by the query region and accumulates the count for those children whose region is contained in the query region.




(i) (ii)

**Fig. 4.** Comparison of (i) running time and (ii) number of I/Os performed when bulk loading the CRB-tree and $kdb$-tree.

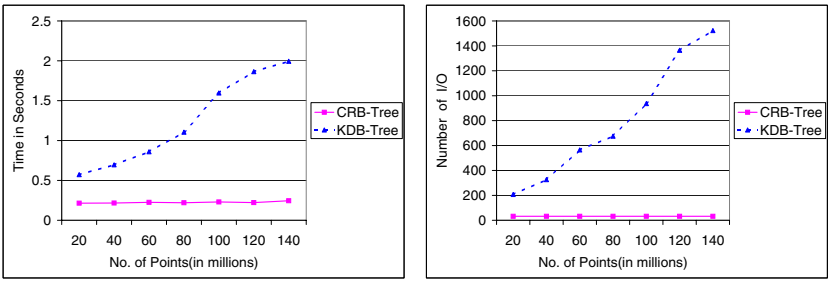
### 4.2 Experiments

We evaluated the performance of the CRB-tree using both synthetic and TIGER/Line data. Below we report both the number of (TPIE) I/Os performed and the wall-clock running time of a set of bulk loading and query experiments. Query bounds are averages over 100 queries with the buffer cache being flushed between queries. All our experiments were performed on a Dell PowerEdge 2400 workstation with a 500MHZ PIII processor and 128MB of main memory, running FREEBSD 4.3. Physically the machine had 1GB of main memory, but to simulate a real multi-user database environment we restricted the main memory usage to 128MB. Furthermore, TPIE was configured to use a maximum of 80MB, leaving the rest of the memory to the operating system. The external memory consisted of a RAID0 disk array of four 36GB SCSI disks (IBM DDYS T36950M).

*Uniformly distributed points.* Our first set of experiments were performed on uniformly distributed points in the range $[0, 10^9] \times [0, 10^9]$ (points were generated by independently choosing a random value for the $x$ and $y$ coordinates). The experiments were performed using data sets sizes ranging from 20 to 140 million points. For each query, we choose a random square with an area equal to 1% of the area of the bounding box of the data set.

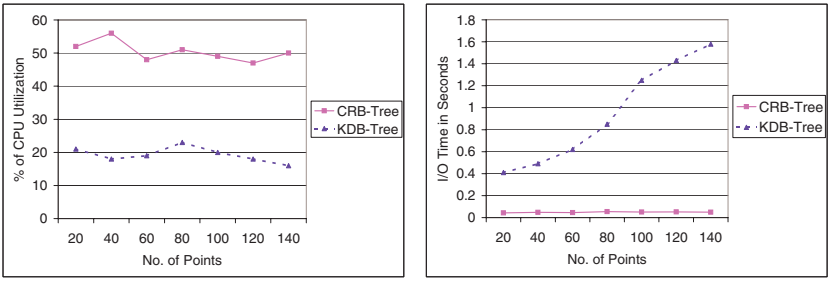
Figure 4 shows the number of I/Os and time taken by the bulk-loading algorithm. The *bulk-loading* time for the CRB-tree is 1.5–2.5 times slower than the $kdB$-tree and as we can see, this is mainly because the number of blocks in the CRB-tree is 3–4 times larger than $kdB$-tree, hence the CRB-tree algorithm performs more I/Os than the $kdB$-tree algorithm.

Figure 5 shows the number of I/Os and time taken by the query process. The *query time* of the CRB-tree is almost independent of the dataset size and significantly lower than the query time of the $kdB$-tree. For the datasets sizes used in the experiments, the height of the CRB-tree ($T$ and $\Psi$) is either 2 or 3. Thus the total number of I/Os performed by the query is at most $6(2*3-1) = 30$. This explains the fact that the query time remains almost constant in these experiments. Since the number of nodes visited by the $kdB$-tree query algorithm increases with increase in data size (it varies as $\sqrt{n}$ in worst case), the query time (I/Os performed) increases significantly as $N$ varies from 20 to 140 million.




(i) (ii)

**Fig. 5.** Comparison of (i) running time and (ii) number of I/Os performed when querying the CRB-tree and $kdB$-tree.




(i) (ii)

**Fig. 6.** Comparison of (i) percentage of CPU calculations and (ii) time to perform I/Os(I/O time) of the query algorithm for CRB-tree and $kdB$ tree.

From Figure 5, we can see that the speedup ratio between CRB-tree and $kdB$-tree is significantly higher for number of I/Os (Figure 5(i)) compared to that of the of execution time (Figure 5(ii)). The reason for this is as follows: The total execution time is composed of three components: (1) user CPU time, (2) I/O time (time spent in performing I/Os) and (3) kernel CPU time. The CRB-tree query process spends a significant time in CPU calculations (because of lots of bit operations) compared to $kdB$-tree query. Figure 6(i) shows the percentage of time spent in CPU calculations for both the CRB-tree and $kdB$-tree query processes. Figure 6(ii) shows the comparison of I/O time of the query processes. As we can see, the speedup of I/O time is almost similar to the speedup of the number of I/Os of Figure 5(ii).

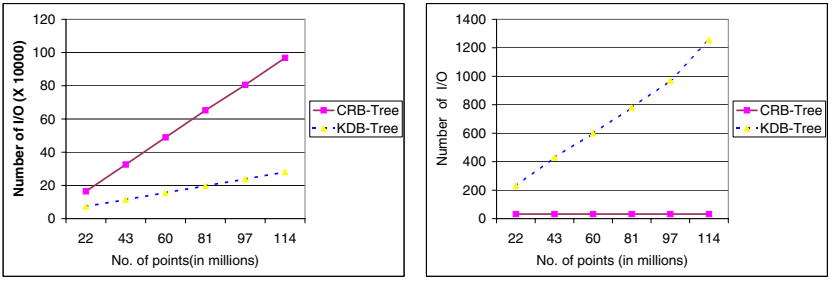
*TIGER/Line data.* We used the TIGER/Line data set from the US Bureau of the Census [19], which is one of the standard benchmark datasets used in spatial databases. The TIGER/Line'97 distribution we used consists of six CD-ROMs of data corresponding to six regions of the United States. We performed experiments with six point datasets, corresponding to the data on CD-ROM 1 through i, for $1 \leq i \leq 6$. The number of points in each of these data sets is shown in Figure 7.

Figure 8 shows the result of bulk loading and query experiments with the TIGER/Line datasets. Since the bulk loading time is independent of the characteristics of the data sets, the bulk loading results are similar to the results we obtained with uniformly distributed points. In the query experiments we again used a randomly placed query square with an

| | CD1 | CD1-2 | CD1-3 | CD1-4 | CD1-5 | CD1-6 |
|---|---|---|---|---|---|---|
| Number of points(in millions) | 22 | 44 | 60 | 81 | 97 | 114 |
| Size(in MB) | 641 | 1238 | 1698 | 2319 | 2789 | 3297 |

**Fig. 7.** Points data sets extracted from TIGER/Line'97

area equal to 1% of the area of the bounding box of the data set. The query performance of CRB-trees is similar to that of uniformly distributed points (the base B-tree $T$ is also of height 3 in these experiments). The query time of the $kdB$-tree on the other hand, increases significantly with increase in dataset size.



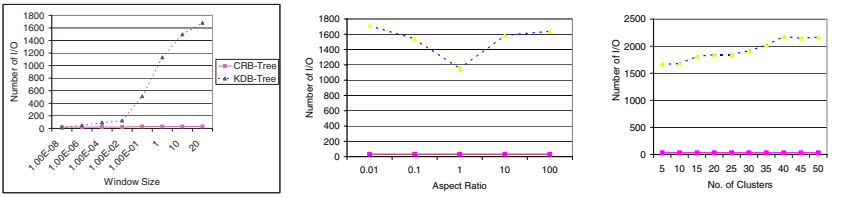

(i) (ii)

**Fig. 8.** Comparison of the number of I/Os performed when (i) bulk loading the CRB-tree and $kdB$-tree and (ii) querying the CRB-tree and $kdB$-tree using TIGER/Line datasets (number of points in millions).

Next we investigated the effect of the query rectangle characteristic on query performance. The experiments were performed using the largest data set of TIGER/LINE. First we performed query experiments with query squares of different sizes. The results of these experiments are shown in Figure 9(i), where the size of the query square is characterized by the ratio of its area to the area of the bounding box of the input points. The size of query square is varied from $10^{-8}\%$ to $20\%$. As it can be seen, the query time of the $kdB$-tree increases rapidly with increasing window sizes. The query time of the CRB-tree on the other hand is almost constant.

Next we performed query experiments with query rectangles instead of query squares. The results of our experiments with rectangles of varying aspect ratio (the ratio between the length and the breadth of the rectangle) are shown in Figure 9(ii). The area of the query rectangle is fixed at 1% of the area of the bounding box of the input dataset while the aspect ratio is varied from 0.01 to 100. As it can be seen, the query time of the $kdB$-tree increases slightly as the query rectangle becomes "skinny" (high aspect ratio or low aspect ratio). The reason for this is that the $kdB$-tree consists of alternating splits along the $x$ and $y$ dimensions and hence a skinny rectangle intersects more nodes of the $kdB$-tree than a square of the same area. As expected, the query time for the CRB-tree is almost constant.

*Clustered data.* Finally, in order to further investigate the influence of the input data distribution on the query performance of the two structures, we performed experiments




(i) (ii) (iii)

**Fig. 9.** Comparison of number of I/Os performed when querying the CRB-tree and $kdB$-tree using the largest TIGER/Line data set and (i) varying the size of query square and (ii) varying the aspect ratio of query rectangle. (iii) shows the comparison of number of I/Os performed during query on synthetic clustered datasets

.

with artificial clustered datasets. The datasets consists of 150 million points distributed evenly among $k$ clusters, where each cluster is generated by uniformly distributing the points on a randomly oriented ellipse of length $4 \times 10^8$ and width $10^4$ centered at $(5 \times 10^8, 5 \times 10^8)$. Figure 9(iii) shows the results of experiments when $k$ is varied from 5 to 50. As previously, the CRB-tree performance is almost constant. Note that the CRB-tree query performance does not depend on whether the input data is uniform or skewed, since the number of I/Os performed by the CRB-tree query procedure, depends only on the height of the tree and not on the distribution of input data.

*Experimental conclusions.* The overall conclusions of our experiments is that while the CRB-tree use 3–4 times more space than the $kdB$-tree and takes 1.5–2.5 times longer to bulk load than a $kdB$-tree, the query performance of the CRB-tree is much better than that of the $kdB$-tree. For a data set with around 100 million points, the CRB-tree query time is 8–10 times faster than $kdB$-tree query time. Furthermore, the query time of the CRB-tree depends only on the height of the tree ($\log_B n$). Thus it is independent of the distribution of the input points and query characteristics, and almost constant for the range of data set sizes used in our experimentation. The query time of the $kdB$-tree on the other hand, depends significantly on the size of the input dataset and the size of the query rectangle. To a lesser extent the query time of the $kdB$-tree also depends on the aspect ratio of the query window and the input point distribution.

**Acknowledgments.** The authors thank Sariel Har-Peled for useful discussions and Octavian Procopiuc for answering numerous questions related to the TPIE system.

## References

1. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

2. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.
3. L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.
4. L. Arge, O. Procopiuc, and J. S. Vitter. Implementing I/O-efficient data structures using TPIE. In *Proc. 10th Annual European Symposium on Algorithms*, pages 88–100, 2002.
5. L. Arge and J. Vahrenhold. I/O efficient dynamic planar point location. In *Proc. ACM Symp. on Computational Geometry*, pages 191–200, 2000.
6. C. Y. Chan and Y. E. Ioannidis. Hierarchical cubes for range-sum queries. In *Proc. of 25th International Conference on Very Large DataBases*, pages 675–686, 1999.
7. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, June 1988.
8. B. Chazelle. Lower bounds for orthogonal range searching, II: The arithmetic model. *J. ACM*, 37:439–463, 1990.
9. H. Edelsbrunner and M. H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14(3):124–128, 1982.
10. V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.
11. S. Geffner, D. Agarwal, and A. E. Abbadi. The dynamic datacube. In *Proc of Intl. Conference on Extending Database Technology*, pages 237–253, 2000.
12. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. of ACM SIGMOD Intl. conference on Management of Data*, pages 205–216, 1996.
13. J. Kim, S. Kang, and M. Kim. Effective temporal aggregation using point-based trees. In *Database and Expert Systems Applications*, pages 1018–1030, 1999.
14. N. Kline and R. T. Snodgrass. Computing temporal aggregates. In *Proc. of Intl conference on Data Engineering*, pages 222–231, 1995.
15. S. Lee, W. Ling, and H. Li. Hierarchical compact cubes for range-max queries. In *Proc of 26th International Conference on Very Large DataBases*, pages 232–241, 2000.
16. J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 725–764. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
17. J. Robinson. The k-d-b tree: A search structure for large multidimensional dynamic indices. In *Proc. of SIGMOD Conference on Management of Data*, pages 10–18, 1981.
18. Y. Tao, D. Papadias, and J. Zhang. Aggregate processing of planar points. In *Extending Database Technology*, pages 682–700, 2002.
19. *TIGER/Line*$^{\text{TM}}$ *Files, 1997 Technical Documentation*. Washington, DC, September 1998.
20. D. E. Vengroff. A transparent parallel I/O environment. In *Proc. DAGS Symposium on Parallel Computation*, 1994.
21. J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of the 17th International Conference on Data Engineering*, pages 51–60, 2001.
22. D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *Proc. Principles Of Database Systems*, pages 237–245, 2001.

# Optimal Range Max Datacube for Fixed Dimensions*

Chung Keung Poon

Dept. of Computer Science, City U. of Hong Kong, China
ckpoon@cs.cityu.edu.hk

**Abstract.** We present a new data structure to support orthogonal range max queries on a datacube. For a $d$-dimensional datacube with size $n$ in each dimension where $d \leq c_3 \log\log n / \log(\log^* n)$, our structure requires $O(c_1^d)$ query time and $O((c_2 n)^d)$ storage where $c_1$, $c_2$ and $c_3$ are constants independent of $d$ and $n$; and $\log^* n$ is the minimum number of repeated logarithms it takes to reduce the value $n$ to at most 2. Hence our data structure is asymptotically optimal when $d$ is fixed, i.e., a constant independent of $n$.

## 1 Introduction

On-line Analytical Processing (OLAP) [11] is concerned with the analysis of huge volumes of data collected in a data warehouse. The data is usually modelled as a multidimensional data cube ([19,13,1]). In particular, a relation with $d+1$ attributes can be viewed as a $d$-dimensional array by designating one attribute as *measure attribute* and the remaining $d$ attributes as *functional attributes*. A tuple in the relation then corresponds to a value located in a particular cell of the array.

A common operation in OLAP systems is that of orthogonal range queries. As an example, suppose we have a relation called `employees` that has attributes: `salary`, `age` and `year`. It can be viewed as a two-dimensional array with dimensions defined by `age` and `year` while the attribute `salary` is chosen as the measure attribute. One might ask for the maximum salary earned by an employee of the age group 20 - 29 between year 1997 and 2000. This can be expressed as the following SQL-like statement:

```
SELECT MAX(salary) FROM employee
WHERE (age>=20) AND (age<=29) AND
      (year>=1997) AND (year<=2000).
```

In general, an orthogonal range query asks for the value of

$$f(R_1, R_2, \ldots, R_d)$$

* This research was fully supported by a grant from the Research Grants Council of the Hong Kong SAR, China [Project No. 9040692 (RGC Ref. No. CityU 1071/02E)].

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 158–172, 2003.

which stands for the application of a function $f$ on the set of values lying within the orthogonal region $R_1 \times R_2 \times \cdots \times R_d$ in $d$-space; and $f$ can be COUNT, SUM, AVERAGE, MAX, MEDIAN, etc.

As OLAP systems are typically used for interactive exploration of the datacube, fast response time is essential. One way to achieve that is to preprocess the datacube and build certain data structure so that subsequent queries can be answered efficiently.

### 1.1 Previous Work

There has been a whole wealth of research in computational geometry dedicated to the study of orthogonal range queries. Most of the best upper bounds typically requires $O(\log^{d-c} N)$ query time and $O(N \log^{d-c'} N)$ storage where $N$ is the number of data points and $c$, $c'$ are positive constants $\leq 2$, see Chazelle [6] and Alstrup et al [2] for example. A central idea in these results is the multi-dimensional divide-and-conquer technique ([4]) which allows one to construct $d$-dimensional data structures from $(d-1)$-dimensional ones with a logarithmic blow-up factor in both the query and storage complexities. There are also lower bounds ([7,8]) indicating the optimality (or near optimality) of these results under certain reasonable computation models.

It was noted (e.g., in [21,20]) that the logarithmic blow-up in query time and storage per dimension might be a problem in OLAP applications. Given the known lower bounds mentioned above, it seems difficult, if not impossible, to build data structures to support efficient OLAP queries. On the other hand, empirical observation suggests that OLAP datacubes are often clustered [12]. Therefore, one may organize the datacube as a collection of densely populated multidimensional arrays and construct efficient range query data structure for each of them separately. Hence, [21,20] initiated the study of orthogonal range queries on dense multidimensional arrays and strived for bringing down the blow-up factor to constant per dimension.

Suppose the array has $d$ dimensions and size $n$ in each dimension. Under the dense array assumption, the number of populated cells is $N = \Theta(n^d)$. Direct evaluation of an orthogonal range query over the array requires examining all the individual values in the region and hence $O(n^d)$ time in the worst case. One can also pre-compute and store all the $O(n^{2d})$ possible answers so that any query takes constant time. Clearly both are impractical solutions. Therefore, the crux of the problem is to design a data structure with constant query time and $O(n^d)$ storage simultaneously.

For range sum queries, Ho et al [21] proposed the Prefix Sum Cube which has query time $O(2^d)$ and space $O(n^d)$. This is optimal for a fixed number of dimensions. However, the update time is expensive, being $O(n^d)$ in the worst case. Geffner et al [17,18] reported a substantial reduction of update complexity to $O(n^{d/2})$ using the Relative Prefix Sum Cube. Further techniques for improving or trading among query time, update time and space have been investigated in [5,16,27,26,10]. On the more theoretical side, Fredman [14] gave a tree structure that supports range sum queries and updates in $O((c_1 \log n)^d)$ time using

$O((c_2 n)^d)$ storage for some constants $c_1$ and $c_2$. There are also other research works on various issues besides the complexities, such as methods for combining different construction strategies in different dimensions ([28]) and providing query answers progressively ([23]).

For range max queries, Ho et al [21] studied a quad-tree-like structure which takes $O(n^d)$ storage. In the worst case, it answers queries in $O(\log n)$ time in the 1-dimensional case and $\Omega(n^{d-1})$ time for $d$-dimensional case. Ho et al [20] then investigated various techniques to improve the average query time. Kim et al [22] introduced the concept of *maximal cover* and evaluated their techniques experimentally. Lee et al [24] proposed the Hierarchical Compact Cube (HCC) which requires $O(n^d)$ space and has constant expected query cost assuming certain random distribution of data and query regions. It is also a quad-tree-like structure. Unfortunately, for certain query regions, in particular those regions with a long and thin shape that partially intersect many small subcubes, HCC still inherits the $\Omega(n^{d-1})$ query complexities typical of a quad-tree structure.

The worst case complexities for the more general problem of semigroup sum queries were first studied in [29]. A semigroup sum operator is an associative operator that may not have an inverse operator. The usual operators, SUM, COUNT, MAX, MIN, are all semigroup operators. (Thus, we define group operators, such as SUM and COUNT, as special cases of semigroup operators.) Yao [29] showed that the 1-dimensional case can be solved by a structure with $s$ (where $s \geq n$) storage and $O(\alpha(s, n))$ query time. (The function $\alpha(s, n)$ is known as the *functional inverse of Ackermann's function*. It is a decreasing function of $s$ and a very slow growing function of $n$.) His result was extended to the $d$-dimensional case by Chazelle and Rosenberg [9] who designed a structure with $s^d$ storage and $O(\alpha^d(s, n))$ query time. This was further generalized to the dynamic case in [25]: For user chosen parameters, $L$ (in $\{1, \ldots, \log n\}$) and $s$ (multiples of $n$), the general formula for the query, update and storage complexities are $O(\alpha^d(s, n)L^d)$, $O(\alpha^d(s, n)L^{2d}n^{d/L})$ and $O(s^d)$ space respectively. Whether there exists a structure in which the query and storage complexities increase by a constant factor independent of $n$ per dimension was left as an open problem.

Note that Yao [29] proved a lower bound showing that the 1-dimensional structure in [29,9,25] is optimal under an arithmetic computation model. In particular, the lower bound applies to any *oblivious* structure, i.e., one in which the construction is independent of the values stored. (More on oblivious structure will be explained in Section 2. For the time being, we just point out that all the constructions in [29,9,25] are oblivious structures.) Therefore, it seems that one must look for non-oblivious data structures. In fact, for range max queries, Gabow et al. [15] presented a (non-oblivious) 1-dimensional structure with $O(1)$ query time and $O(n)$ space. Recently, Bender and Farach-Colton [3] designed another optimal non-oblivious data structure for range max queries. Unfortunately, all these techniques do not scale up to higher dimensions: There are no known techniques that extend arbitrary (non-oblivious) $(d-1)$-dimensional range query

structures to $d$-dimension ones without paying a logarithmic blow-up in query and space complexities.

### 1.2 Our Contribution

In this paper, we solve the open problem for the special case of range max queries by presenting a data structure that has $O(c_1^d)$ query time and $O((c_2 n)^d)$ storage for $d \leq c_3 \log\log n / \log(\log^* n)$ and some constants $c_1$, $c_2$ and $c_3$ independent of $d$ and $n$. (The function $\log^* n$ is defined as the minimum number of repeated logarithms it takes to reduce the value $n$ to at most 2.) Hence it is optimal when $d$ is a constant independent of $n$. The result is achieved by first designing an alternative optimal 1-dimensional range max structure, and then extending it to higher dimensions using a modification of the *cross-product* techniques described in [9,28,25].

To simplify the exposition, we only describe the construction of static structures and leave the discussion of dynamic structures to the full paper. In the next section, we state our notations and machine model, and briefly review some of the basic techniques used in previous works. In Section 3 and 4, we describe our optimal 1- and 2-dimensional range max structures. These are generalized to $d$ dimensions in Section 5. We then conclude the paper with some open problems in Section 6.

## 2 Overview

### 2.1 Notations

The original multidimensional datacube (or array) is denoted as $A$. We call it the *base* array. We assume $A$ has $d$ dimensions and size $n$ in each dimension. Note, however, that our technique works for arrays with different sizes in different dimensions.

Let $i$, $j$ and $a$ be integers. We denote by $i_{(a)}$ the value $\lfloor i/a \rfloor a$. We call an interval $[i..j]$ an $a$-interval if $i$ is a multiple of $a$ and $j = i + a - 1$ (or $j = n - 1$ if $i + a > n$).

Since we will only be concerned with orthogonal range max queries, we represent such a query, $\text{MAX}([i_1..j_1], [i_2..j_2], \ldots, [i_d..j_d])$, simply by the query region, $[i_1..j_1] \times [i_2..j_2] \times \cdots \times [i_d..j_d]$. The array to be queried is understood to be $A$ unless otherwise stated. Similarly, we represent an entry or a subarray of $A$ using the notation: $A[i_1..j_1][i_2..j_2] \cdots [i_d..j_d]$.

### 2.2 The Model

Let us define our computation model. We assume each storage unit (called *word*) is capable of storing a value or an index in a dimension. Hence a word contains at least $b \log n$ bits for some constant $b \geq 1$. The idea of storing an index in a word is not new. Various previous works, such as [20,24], require storing indices in

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $P_0(=X)$ | 89 | 35 | 18 | 11 | 13 | 71 | 24 | 67 |
| $P_1$ | 89 | 35 | 18 | 18 | 71 | 71 | 24 | 67 |
| $P_2$ | 89 | 35 | 18 | 11 | 13 | 71 | 71 | 71 |

**Fig. 1.** Content of $APM_{1,\log n}(X)$ for $n = 8$

a word. Thus, storing a $d$-dimensional index would require $O(d)$ words. Storing the multidimensional array $A$ would require $O(n^d)$ words since we need not store the indices. We charge one time step for each access to a word. Time for index calculation and applications of the max operator are not counted. We ignore issues such as blocking factor of disk and caching as physical design issues.

### 2.3 Review of Techniques

Our basic techniques come from [25]. Below, we briefly review the required concepts.

An $(l, h)$-APM structure for an array $X[0..n-1]$, denoted $APM_{l,h}(X)$, is a collection of $h$ arrays, $P_0, P_1, \ldots, P_{h-1}$, each of length $n$. For $0 \leq w < h$, $P_w$ is defined as:

$$P_w[i] = \begin{cases} \max X[i_{(2^w l)}..i] & \text{if } \lfloor i/(2^w l) \rfloor \text{ is odd,} \\ \max X[i..i_{(2^w l)} + 2^w l - 1] & \text{if } \lfloor i/(2^w l) \rfloor \text{ is even.} \end{cases}$$

In other words, $P_w$ is composed of alternating suffix max arrays and prefix max arrays of $X$. (Our notation and formula here are slightly different from that in [25] since some small improvements have been made. However, this does not change the asymptotic behaviour of the structure.)

Note that the structure $APM_{1,\log n}(X)$ is able to answer any range max query on $X$ in 2 steps as follows. Given a query $[i..j]$, we first compute $k = MSB(i \oplus j)$ where $i \oplus j$ is the bitwise exclusive-or of the binary representations of $i$ and $j$, and $MSB(x)$ is the bit position of the most significant 1-bit in a binary number $x$. Then the maximum value in $X[i..j]$ is simply the maximum between $P_k[i]$ and $P_k[j]$. For example, the table in Figure 1 shows the content of $APM_{1,\log n}(X)$ for $n = 8$. Note that by definition $P_0$ is the same as $X$.

Since $MSB(4, 5) = 0$, the answer to query $[4..5]$ is the maximum of $P_0[4] = 13$ and $P_0[5] = 71$. Similarly, the maximum of $X[1..6]$ is the maximum of $P_2[1] = 35$ and $P_2[6] = 71$, since $MSB(1, 6) = 2$.

Furthermore, it was shown in [25] that a number of APM structures with carefully chosen parameters can be combined to give a (1-dimensional) range max structure with space $O(n \log^* n)$ and query time 4. More precisely, for $n \leq g(k)$, there is a structure of size $kn$ that can answer any range max queries in 4 steps, where the function $g(k)$ is defined as:

$$g(1) = 4$$
$$g(2) = 10$$

| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $g(k)$ | 4 | 10 | 32 | 64 | 10240 | $2^{37}$ | $2^{70}$ |

**Fig. 2.** Values of $g(k)$

$$g(3) = 32$$
$$g(k) = g(k-3) \cdot 2^{g(k-3)} \qquad \text{for } k > 3.$$

The value of $g(k)$ as $k$ increases is shown in Figure 2.

Hence for an array $A$ of length $n$, the required storage is $n\gamma(n)$ where $\gamma(n)$ is the smallest $k$ such that $n \leq g(k)$. It can be shown that $\gamma(n) = O(\log^* n)$. We will call this the *log-star* structure.

Besides having a fast query time and nearly optimal space complexity (on an arithmetic model), another advantage of the log-star structure is that it is oblivious. A structure is *oblivious* if its construction and the places to be probed during a query are independent of the values in the array.

Finally, by cross-product technique, we refer to a general idea for constructing higher dimensional structures from lower dimensional oblivious structures. Here we will just give the gist of idea. Let $A$ be a 2-dimensional array. We construct a 1-dimensional oblivious range max structure on each row of $A$ to produce another array $B$ where one row of $B$ corresponds to one structure. Then we construct a 1-dimensional oblivious range max structure for each column of $B$ (using possibly a different construction strategy) to produce an array $C$. Suppose we apply the log-star construction, which is oblivious. Then it is not hard to see that the maximum of a 2-dimensional region in $A$ is equivalent to the maximum in 4 vertical regions in $B$, which in turn, is equivalent to the maximum of 16 points in $C$. Note that the resultant structure is also oblivious. Thus we can apply the same technique repeatedly to obtain a $d$-dimensional structure.

## 3 One-Dimensional Queries

Throughout this section, $d = 1$. Therefore, $A$ is a linear array of length $n$. Let $m = c \log^* n$ where $c$ is some constant to be chosen. For simplicity, assume $n$ is a multiple of $m$.

### 3.1 Construction and Storage

Our structure consists of two parts, the *macro-structure* and the *micro-structure*. The macro-structure is stored in an array $X$ of length $O(n)$ constructed as follows. Define $A_m$ as the array of size $n/m$ such that for $i = 0$ to $n/m - 1$,

$$A_m[i] = \max A[im..(i+1)m - 1].$$

We construct a range max structure on $A_m$ with query time 4 and storage $O((n/m)\log^*(n/m)) = O(n)$ using the log-star technique. The resultant structure is stored in $X$. (Note that $A_m$ is not stored.) The construction time is linear to the size of $X$.

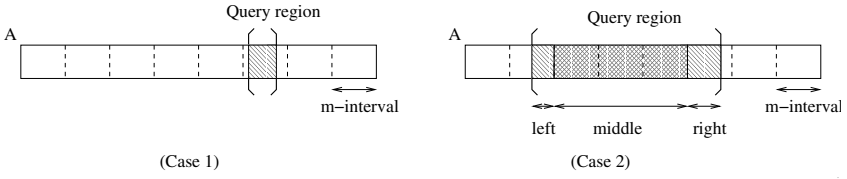



**Fig. 3.** One-dimensional queries

The micro-structure consists of an array $Y$ of length $n$. Each array position $i$ in $A$ will be associated with the word $Y[i]$. Into this word, we pack the $m$ indices to the answers of the $m$ query regions: $[i..i], [i..i+2], \ldots, [i..i+m-1]$. Note that the index to an answer has offset at most $m-1$ from position $i$. Therefore, we just store the offset from $i$. This requires only $\log m$ bits. In total, we pack $m \log m$ bits into $Y[i]$. This is feasible since $m \log m = (c \log^* n) \log(c \log^* n) \leq b \log n$ for sufficiently large $n$. The construction takes $O(nm)$ time to complete.

The whole structure requires $O(n)$ space. For concreteness, we assume $b$ and $c$ are chosen such that the macro- and micro-structures require exactly $n$ words each. Hence the whole structure (including $A$) requires $3n$ storage.

As an example, consider $n = 64$ and take $b = 2$. Then word size is $b \log n = 12$. According to Figure 2, the log-star structure requires $4n$ storage. Choosing $m = 4$, the macro-structure requires no more than $(n/m)m = n$ storage. As for the micro-structure, we store in each word of $Y$ $m$ offsets each ranging from 0 to $m-1$. This requires $4 \log 4 = 8$ bits in total and these can be packed into one word.

Repeating the above calculation for $n = 20000$, the word size is $b \log n = 36$. Notice from Figure 2 that the log-star structure requires $6n$ storage. Hence we choose $m = 6$ and pack $6 \log 6 = 18$ bits into a word which is, again, feasible.

### 3.2 Query

To answer a query $[i..j]$, we consider two cases, see Figure 3.

*(Case 1)* $[i..j]$ lies within an $m$-interval.
We make use of the micro-structure, $Y$, to compute the answer. We extract the $(j-i)$-th offset, say $k$, from the word $Y[i]$. Then the answer to the query is in $A[i+k]$. This requires 2 steps, one for reading $Y$ and one for $A$. (The extraction of $k$ from $Y[i]$ and the addition to $i$ are not counted towards the query complexity.)

*(Case 2)* $[i..j]$ spans across at least two $m$-intervals.
We separate the query region into (at most) three parts: $[i..i'-1]$, $[i'..j'-1]$ and $[j'..j]$ where $i' = (i-1)_{(m)} + m$ and $j' = (j+1)_{(m)}$. The middle part consists of complete $m$-intervals and the maximum can be found by the macro-structure, $X$, in 4 steps. The maximum of the left and right part can be found by the micro-structure, $Y$, in 2 steps each as described in (Case 1). Hence a query requires at most 8 steps.

## 4 Two-Dimensional Queries

Note that the macro-structure of the above 1-dimensional structure is oblivious in the technical sense as defined in [25]. The micro-structure is also "oblivious" in the sense that the offset to be fetched is solely determined by the query region. Therefore, one might think that a 2-dimensional structure can be constructed easily using the cross-product technique. However, remember that the content of a word, say $Y[i]$, in the micro-structure alone is not sufficient to define a value in the base array. It has to be combined with the index of the word, i.e., $i$, in order to recover the index to the base array. This implies that the content in the micro-structure cannot be freely copied from one place to another in the construction of higher dimensional structures. We overcome this problem by modifying the cross-product technique.

### 4.1 Construction and Storage

For each $i \in [0..n-1]$, we construct a macro-structure for $A[i][0..n-1]$. Let them be stored in an array $B[0..n-1][0..n-1]$. Then for each $j \in [0..n-1]$, we construct a 1-dimensional range max structure (including both macro- and micro-structures) for $B[0..n-1][j]$. Let the resultant structures be stored in $X[0..2n-1][0..n-1]$.

Next, we construct a macro-structure for $A[0..n-1][j]$ for each $j \in [0..n-1]$ and let them be stored in $C[0..n-1][0..n-1]$. Then we construct a micro-structure for $C[i][0..n-1]$ for each $i \in [0..n-1]$. Let them be stored in $Y[0..n-1][0..n-1]$.

Finally, we construct a 2-dimensional micro-structure $Z[0..n-1][0..n-1]$ such that $Z[i][j]$ stores the $m^2$ answers to the $m^2$ query regions: $[i..i+x] \times [j..j+y]$ where $0 \leq x, y < m$. Again, we store the offsets of the position of the answer from $i$ and $j$ respectively. Thus, each 2-dimensional offset requires $2 \log m$ bits. This is feasible as long as $m^2 \log(m^2) \leq b \log n$.

In total, we need to store the arrays, $A, B, C, X, Y$ and $Z$, using a total of $7n^2$ storage. The construction time is $O(n^2 m + (n/m)^2 m^4) = O((nm)^2)$.

### 4.2 Query

To answer a query $[i_1..j_1] \times [i_2..j_2]$, we consider three cases. See Figure 4.

*(Case 1)* $[i_2..j_2]$ spans across more than one $m$-interval in dimension 2. We divide the query into at most three parts: $[i_1..j_1] \times [i_2..i'_2 - 1]$, $[i_1..j_1] \times [i'_2..j'_2 - 1]$ and $[i_1..j_1] \times [j'_2..j_2]$ where $i'_2 = (i_2 - 1)_{(m)} + m$ and $j'_2 = (j_2 + 1)_{(m)}$.

For the middle part, we will make four 1-dimensional queries $[i_1..j_1]$ to row $p_1, p_2, p_3$ and $p_4$ of $X$ where $p_1, p_2, p_3$ and $p_4$ are positions along dimension 2 determined by the macro-structure for a column of $A$ and the query range $[i'_2..j'_2 - 1]$. This requires $4 \times 8 = 32$ steps. The top and bottom parts are handled as in (Case 2) or (Case 3) below.

*(Case 2)* $[i_2..j_2]$ lies within an $m$-interval in dimension 2 but $[i_1..j_1]$ spans across more than one $m$-interval in dimension 1.

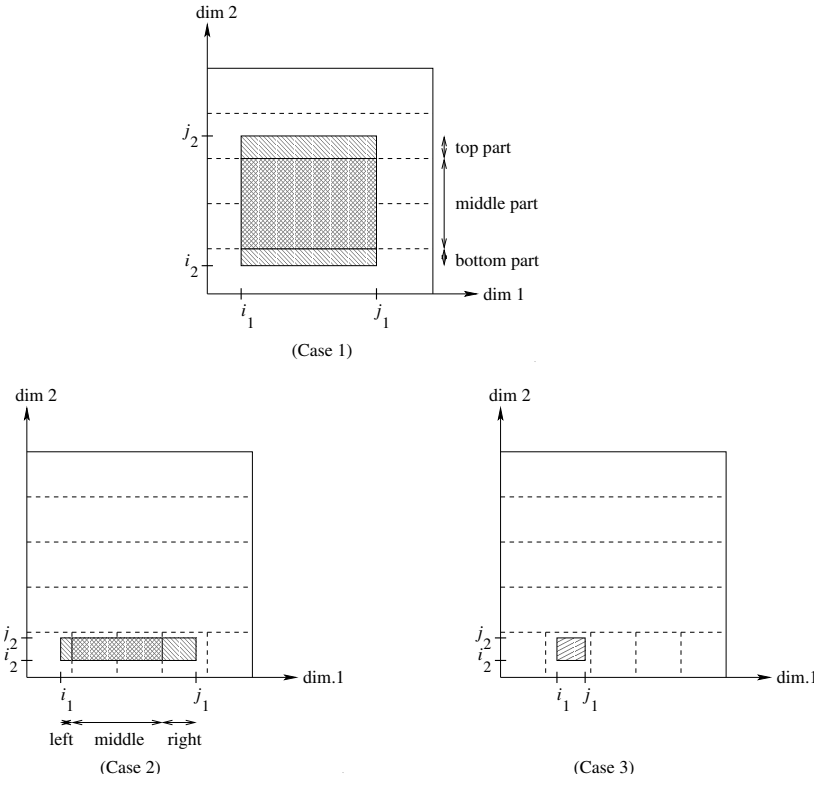



**Fig. 4.** Two-dimensional queries

We divide the query into at most three parts: $[i_1..i'_1 - 1] \times [i_2..j_2]$, $[i'_1..j'_1 - 1] \times [i_2..j_2]$ and $[j'_1..j_1] \times [i_2..j_2]$ where $i'_1 = (i_1 - 1)_{(m)} + m$ and $j'_1 = (j_1 + 1)_{(m)}$.

The middle part is computed by four 1-dimensional queries $[i_2..j_2]$ to column $q_1, q_2, q_3$ and $q_4$ of $Y$ where $q_1, q_2, q_3$ and $q_4$ are positions along dimension 1 determined by the macro-structure for a row of $A$ and $[i'_1..j'_1 - 1]$. Each requires 1 step to look up $Y$ and then 1 step to look up $C$. This requires $4 \times 2 = 8$ steps. The left and right parts are computed as in (Case 3) below.

*(Case 3)* Both $[i_1..j_1]$ and $[i_2..j_2]$ are within an $m$-interval in their respective dimension.

In this case, we simply lookup $Z[i_1][i_2]$ in one step to find the appropriate offsets $x_1$, $x_2$ and then lookup $A[i_1 + x_1][i_2 + x_2]$ for the answer. This requires 2 steps.

Summarizing all 3 cases, we need at most $32 + 2 \times (8 + 2 \times 2) = 56$ steps to compute the answer.

# 5 Extensions to Higher Dimensions

## 5.1 A Function Notation

To facilitate the discussion, we will adopt a function notation (function as in a programming language rather than mathematical functions) to describe our data structure construction and storage requirement.

Our primitive functions are $\mathcal{F}$ and $\mathcal{G}$ which denote the macro- and micro-structure construction respectively. We denote by the expression $B = \mathcal{F}_k(A)$, the application of the macro-structure construction along dimension $k$ of $A$ to yield the structure $B$. That is, for each possible value of $(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_d)$, $B[i_1] \cdots [i_{k-1}][0..n-1][i_{k+1}] \cdots [i_d]$ is a macro-structure for $A[i_1] \cdots [i_{k-1}][0..n-1][i_{k+1}] \cdots [i_d]$. Similarly, $\mathcal{G}_k$ denotes the application of the micro-structure construction along dimension $k$.

Using this notation, the construction of our 1-dimensional structure can be described as a function, denoted $\mathcal{H}^1$, as follows:

structure $\mathcal{H}^1(A)$
begin
    $X = \mathcal{F}_1(A)$
    $Y = \mathcal{G}_1(A)$
    export $(X, Y)$
end

The explicit reference to an array on the left hand side of an equality sign represents explicit storage of that array. For the example above, both $X$ and $Y$ are stored. In fact, they are also the "return value" of the function $\mathcal{H}_1$ as signified by the statement "export $(X, Y)$".

## 5.2 Function Composition

As a second example, the construction of a 2-dimensional structure can be described as the following function $\mathcal{H}^2$:

structure $\mathcal{H}^2(A)$
begin
    $B = \mathcal{F}_2(A)$
    $(X_1, X_2) = \mathcal{H}^1(B)$
    $(C_1, C_2) = \mathcal{H}^1(A)$
    $Y_1 = \mathcal{G}_2(C_1)$, $Y_2 = \mathcal{G}_2(C_2)$
    export $(X_1, X_2, Y_1, Y_2)$
end

Plugging in definition of $\mathcal{H}^1$, we can see that essentially (ignoring issues on what intermediate arrays are stored), $X_1 = \mathcal{F}_1(\mathcal{F}_2(A))$ and $X_2 = \mathcal{G}_1(\mathcal{F}_2(A))$. The meaning of the function compositions $\mathcal{F}_1\mathcal{F}_2$ and $\mathcal{G}_1\mathcal{F}_2$ are quite clear as the "return value" of $\mathcal{F}_2$ is an array. Similarly, $Y_1 = \mathcal{G}_2(\mathcal{F}_1(A))$ is easily understood for the same reason.

However, some explanation on $Y_2 = \mathcal{G}_2(\mathcal{G}_1(A))$ is in order. First, consider the structure $C_2$ returned by $\mathcal{G}_1(A)$. The word $C_2[i][j]$ contains the $m$ offsets (relative to $i$) to the maximums in the $m$ regions $A[i..i][j], A[i..i+1][j], \ldots, A[i..i+m-1][j]$. Applying $\mathcal{G}_2$ on $C_2$, the values referenced by the $x$-th offsets of $C_2[i][j], \ldots, C_2[i][j+y]$ will be compared and the 2-dimensional offsets to the maximum (relative to $(i,j)$) will be stored in $Y_2[i][j]$. This offset will point to the maximum value in $A[i..i+x][j..j+y]$. This holds for each $(x,y) \in [0..m-1] \times [0..m-1]$. Hence $Y_2[i][j]$ will store $m^2$ offsets. Thus $Y_2$ is actually the same as $Z$ in Section 4.

Note that storage of $C_2$ is not necessary since offsets in $Y_2$ refer directly to $A$. However, for the sake of easy analysis, we assume $C_2$ is stored without much effect on our storage complexity.

Finally, we can generalize a sequence of micro-structure compositions, $\mathcal{G}_d \cdots \mathcal{G}_1$, as a construction producing an array in which each word stores $m^d$ $d$-dimensional offsets. This is feasible as long as $(m^d)\log(m^d) \leq \log n$ which is satisfied by the condition $d \leq c_3 \log\log n / \log m$ for some constant $c_3$.

### 5.3 Construction and Storage

Now we are ready to describe the construction of our $d$-dimensional structure $\mathcal{H}^d$:

structure $\mathcal{H}^d(A)$
begin
  $B = \mathcal{F}_d(A)$
  $(X_1, \ldots, X_{2^{d-1}}) = \mathcal{H}^{d-1}(B)$
  $(C_1, \ldots, C_{2^{d-1}}) = \mathcal{H}^{d-1}(A)$
  $Y_1 = \mathcal{G}_d(C_1), \ldots, Y_{2^{d-1}} = \mathcal{G}_d(C_{2^{d-1}})$
  export $(X_1, \ldots, X_{2^{d-1}}, Y_1, \ldots, Y_{2^{d-1}})$
end

First, let us analyze the storage required. Let $s_d n^d$ be the internal storage needed by $\mathcal{H}^d(A)$. The internal storage includes the arrays $B$ and $C_1, \ldots, C_{2^{d-1}}$ and the internal storage for $\mathcal{H}^{d-1}(A)$ and $\mathcal{H}^{d-1}(B)$. Therefore we have the relation:

$$\begin{aligned} s_1 &= 0 \\ s_d &= 2s_{d-1} + 1 + 2^{d-1}. \end{aligned}$$

Besides the internal storage, we need to store the base array $A$ and the exported arrays $(X_1, \ldots, X_{2^d})$ and $(Y_1, \ldots, Y_{2^d})$. Hence the total storage required by a $d$-dimensional structure is $(s_d + 2^d + 1)n^d$.

We claim that $s_d + 2^d + 1 \leq 3^d$ for all $d \geq 1$. This can be proved easily be induction on $d$. When $d = 1$, $s_1 + 2^1 + 1 = 3 \leq 3^1$. Therefore, the claim is true for $d = 1$. Assume the claim is true for some $d-1$ where $d > 1$. Then

$$\begin{aligned} s_d + 2^d + 1 &= (2s_{d-1} + 2^{d-1} + 1) + 2^d + 1 \\ &\leq 2(s_{d-1} + 2^{d-1} + 1) + 2^{d-1} \end{aligned}$$

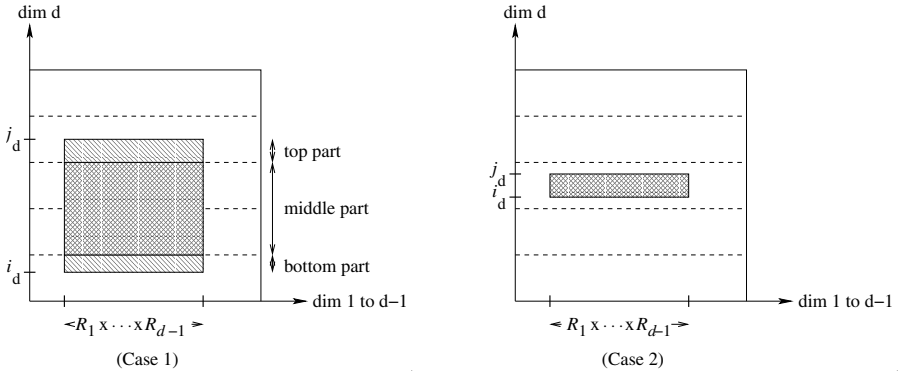



**Fig. 5.** $d$-dimensional queries

$$\leq 2 \cdot 3^{d-1} + 3^{d-1}$$
$$\leq 3^d.$$

Hence the claim follows and our data structure requires $(3n)^d$ storage.

Next, we will analyze the construction time, $P_d(n)$, of $\mathcal{H}^d(A)$. We first claim that the number of exported precomputed ranges in $\mathcal{H}^d(A)$ is $(n+nm)^d$. This is obviously true for $d = 1$. Assume this is true for $d-1$ and consider $\mathcal{H}^d(A)$. The number of precomputed ranges in the $X_i$'s is $n(n+nm)^{d-1}$. The number of precomputed ranges in the $C_i$'s is $(n+nm)^{d-1}$. When constructing $Y_i$ from $C_i$, the number of precomputed ranges increased by a factor of $(n/m) \times m^2 = nm$. Hence the number of exported precomputed ranges in $\mathcal{H}^d(A)$ is $n(n+nm)^{d-1} + (n+nm)^{d-1}nm = (n+nm)^d$. Hence the claim follows.

As for $P_d(n)$, we will show that $P_d(n) \leq O((3nm)^d)$. This is true for $d = 1$ as shown before. Assume this is true for $d-1$ and consider the construction of $\mathcal{H}^d(A)$. Construction of $B$ requires $O(n^{d-1}n)$ time. Construction of $X_i$'s and $C_i$'s requires $2nP_{d-1}(n)$ time. Finally, there are at most $(n+nm)^{d-1}$ exported precomputed ranges in the $C_i$'s by the above claim. Hence the construction of $Y_i$'s require $O((n+nm)^{d-1} \cdot nm)$ time. In total,

$$\begin{aligned} P_d(n) &= O(n^{d-1}n) + 2nP_{d-1}(n) + O((n+nm)^{d-1} \times (nm)) \\ &\leq O(n^d + 2 \cdot 3^{d-1}n^d m^{d-1} + 2^{d-1}n^d m^d) \\ &\leq O(2 \cdot 3^{d-1}n^d m^{d-1} + 3^{d-1}n^d m^d) \\ &\leq O((3nm)^d). \end{aligned}$$

### 5.4 Query

Given a query $R_1 \times \cdots \times R_{d-1} \times [i..j]$ where $R_1, \ldots, R_{d-1}$ are ranges on dimension 1 to $d-1$, we consider two cases, see Figure 5:

*(Case 1)* $[i..j]$ spans across more than one $m$-interval in dimension $d$.
As before, we break it down into at most 3 parts: $R_1 \times \cdots \times R_{d-1} \times [i..i'-1]$,

$R_1 \times \cdots \times R_{d-1} \times [i'..j'-1]$ and $R_1 \times \cdots \times R_{d-1} \times [j'..j]$ where $i' = (i-1)_{(m)} + m$ and $j' = (j+1)_{(m)}$.

Maximum in the middle part is computed by four $(d-1)$-dimensional queries $X_1 \times \ldots \times X_{2^{d-1}}$ on four different positions along dimension $d$ determined by $\mathcal{F}_d$ and $[i'..j'-1]$. The top and bottom parts are handled as in (Case 2) below.

*(Case 2)* $[i..j]$ lies within one $m$-interval in dimension $d$.
We make $t$ 1-dimensional queries $[i..j]$ on $(Y_1, \ldots, Y_{2^{d-1}})$ where $t$ is at most the number of steps required by the $(d-1)$-dimensional query $R_1 \times \cdots \times R_{d-1}$ on $\mathcal{H}^{d-1}(A)$. The results, which are offsets of varying dimensions, are converted back to indices to $(C_1, \ldots, C_{2^{d-1}})$ and the same number $t$ of lookups are made on $(C_1, \ldots, C_{2^{d-1}})$ to read off the values from which the maximum is calculated.

To analyze the time complexity, let $t_d$ be the worst case time needed for a $d$-dimensional query. Then (Case 2) above requires at most $t_d$ queries to $(Y_1, \ldots, Y_{2^{d-1}})$ and the same number of lookups to $(C_1, \ldots, C_{2^{d-1}})$. Hence we have

$$\begin{aligned} t_1 &= 8 \\ t_d &= 4t_{d-1} + 2(2 \times t_{d-1}) \\ &= 8^d. \end{aligned}$$

## 6 Conclusion

In this paper, we proposed a new data structure for supporting range max queries in OLAP which has optimal query time and space requirement for fixed number of dimensions $d$. For simplicity of presentation, we have not optimized the constants $c_1$ and $c_2$ in the query and storage complexities. Also, we can apply the tradeoff technique of Poon [25] to obtain a data structure with update time $O((4L^2 n^{1/L} \log^* n)^d)$ while increasing the query time to $O((3L)^d)$. It remains to design optimal structures for general $d$ dimensional arrays. Another challenging open problem is to design a data structure for general semigroup sum queries with similar performance.

## References

1. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *13th International Conference on Data Engineering*, pages 232–243. IEEE, 1997.
2. S. Alstrup, G.S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science*, pages 198–207. IEEE, 2000.
3. M. Bender and M. Farach-Colton. The LCA problem revisited. In *4th Latin American Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94, Punta del Este, Uruguay, April 2000. Springer-Verlag.
4. J.L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–228, April 1980.

5. C.Y. Chan and Yannis E. Ioannidis. Hierarchical prefix cubes for range-sum queries. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 675–686. Morgan-Kaufmann, 1999.
6. Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, June 1988.
7. Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, April 1990.
8. Bernard Chazelle. Lower bounds for orthogonal range searching: II. the arithmetic model. *Journal of the ACM*, 37(3):439–463, July 1990.
9. Bernard Chazelle and Burton Rosenberg. Computing partial sums in multidimensional arrays. In *5th Annual Symposium on Computational Geometry*, pages 131–139. ACM, 1989.
10. Seok-Ju Chun, Chin-Wan Chung, Ju-Hong Lee, and Lee Seok-Lyong. Dynamic update cube for range-sum queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 521–530, Roma, Italy, 2001.
11. E.F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: an IT mandate. Technical report, E.F. Codd and Associates, 1993.
12. George Colliat. OLAP, relational and multidimensional database systems. *SIGMOD RECORD*, September 1996.
13. The OLAP Council. MD-API the OLAP application program interface version 0.5 specification. Technical report, September 1996.
14. Michael Fredman. A lower bound on the complexity of orthogonal range queries. *Journal of the ACM*, 28(4):696–705, 1981.
15. H. Gabow, J.L. Bentley, and R.E. Tarjan. Scaling and related techniques for goemetry problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 135–143. ACM, 1984.
16. S. Geffner, D. Agrawal, and A. El Abbadi. The dynamic data cube. In *Advances in Database Technology (EDBT'2000)*, volume 1777 of *Lecture Notes in Computer Science*, pages 237–253. Springer-Verlag, March 2000.
17. S. Geffner, D. Agrawal, A. El Abbadi, and T. Smith. Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. In *15th International Conference on Data Engineering*, pages 328–335. IEEE, 1999.
18. S. Geffner, M. Riedewald, D. Agrawal, and A. El Abbadi. Data cubes in dynamic environments. *IEEE Data Engineering Bulletin*, 22(4):31–40, 1999.
19. Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *12th International Conference on Data Engineering*, pages 152–159. IEEE, 1996.
20. Ching-Tien Ho, Rakesh Agrawal, Nimrod Meggido, and Jyh-Jong Tsay. Techniques for speeding up rang-max queries. Technical report, IBM Research Report, April 1997.
21. Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in OLAP data cubes. In *ACM SIGMOD Conference on the Management of Data*, pages 73–88, 1997.
22. Dong Wook Kin, Eun Jung Lee, Myoung Ho Kim, and Yoon Joon Lee. An efficient processing of rnage-MIN/MAX queries over data cube. *Information Scieces*, 112:223–237, 1998.
23. Iosif Lazaridis and Sharad Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *ACM SIGMOD Conference on the Management of Data*, pages 401–412, May 2001.

24. Sin Yeung Lee, Tok Wang Ling, and HuaGang Li. Hierarchical compact cube for range-max queries. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 232–241, Cairo, Egypt, 2000. Morgan-Kaufmann.
25. Chung Keung Poon. Orthogonal range queries in OLAP. In *8th International Conference on Database Theory*, volume 1973 of *Lecture Notes in Computer Science*, pages 361–374, London, Britain, January 2001. Springer-Verlag.
26. M. Riedewald, D. Agrawal, A. El Abbadi, and R. Pajarola. Space-efficient data cubes for dynamic environments. In *2nd International Conference on Data Warehousing and Knowledge Discovery*, volume 1874 of *Lecture Notes in Computer Science*, pages 24–33, London, Britain, September 2000. Springer-Verlag.
27. Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. pCube: update-efficient online aggregation with progressive feedback and error bounds. In *Proceedings of 12th International Conference on Scientific and Statistical Database Management*, pages 95–108. IEEE Computer Society, 2000.
28. Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Flexible data cubes for online aggregation. In *8th International Conference on Database Theory*, volume 1973 of *Lecture Notes in Computer Science*, pages 159–173, London, Britain, January 2001. Springer-Verlag.
29. Andrew Yao. Space-time tradeoff for answering range queries. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 128–136, 1982.

# Processing XML Streams with Deterministic Automata

Todd J. Green[1], Gerome Miklau[2], Makoto Onizuka[3], and Dan Suciu[2]

[1] Xyleme SA, Saint-Cloud, France todd.green@xyleme.com
[2] University of Washington Department of Computer Science
{gerome,suciu}@cs.washington.edu
[3] NTT Cyber Space Laboratories, NTT Corporation, oni@acm.org

**Abstract.** We consider the problem of evaluating a large number of XPath expressions on an XML stream. Our main contribution consists in showing that Deterministic Finite Automata (DFA) can be used effectively for this problem: in our experiments we achieve a throughput of about 5.4MB/s, independent of the number of XPath expressions (up to 1,000,000 in our tests). The major problem we face is that of the size of the DFA. Since the number of states grows exponentially with the number of XPath expressions, it was previously believed that DFAs cannot be used to process large sets of expressions. We make a theoretical analysis of the number of states in the DFA resulting from XPath expressions, and consider both the case when it is constructed eagerly, and when it is constructed lazily. Our analysis indicates that, when the automaton is constructed lazily, and under certain assumptions about the structure of the input XML data, the number of states in the lazy DFA is manageable. We also validate experimentally our findings, on both synthetic and real XML data sets.

## 1 Introduction

Several applications of XML stream processing have emerged recently: content-based XML routing [24], selective dissemination of information (SDI) [3,6,9], continuous queries [7], and processing of scientific data stored in large XML files [13,25,19]. They commonly need to process large numbers of XPath expressions (say 10,000 to 1,000,000), on continuous XML streams, at network speed.

For illustration, consider XML Routing [24]. Here a network of *XML routers* forwards a continuous stream of XML packets from data producers to consumers. A router forwards each XML packet it receives to a subset of its output links (other routers or clients). Forwarding decisions are made by evaluating a large number of XPath filters, corresponding to clients' subscription queries, on the stream of XML packets. Data processing is minimal: there is no need for the router to have an internal representation of the packet, or to buffer the packet after it has forwarded it. Performance, however, is critical, and [24] reports very poor performance with publicly-available tools.

Our contribution here is to show that the lazy Deterministic Finite Automata (DFA) can be used effectively to process large numbers of XPath expressions, at

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 173–189, 2003.

guaranteed throughput. The idea is to convert all XPath expressions into a single DFA, then evaluate it on the input XML stream. DFAs are the most efficient means to process XPath expressions: in our experiments we measured a sustained throughput of about 5.4MB/s for arbitrary numbers of XPath expressions (up to 1,000,000 in our tests), outperforming previous techniques [3] by factors up to 10,000. But DFAs were thought impossible to use when the number of XPath expressions is large, because the size of the DFA grows exponentially with that number. We analyze here theoretically the number of states in the DFA for XPath expressions, and consider both the case when the DFA is constructed eagerly, and when it is constructed lazily. For the eager DFA, we show that the number of label wild cards (denoted $*$ in XPath) is the only source of exponential growth in the case of a single, linear XPath expression. This number, however, is in general small in practice, and hence is of little concern. For multiple XPath expressions, we show that the number of expression containing descendant axis (denoted // in XPath) is another, much more significant source of exponential growth. This makes eager DFAs prohibitive in practice. For the lazy DFA, however, we prove an upper bound on their size that is independent of the number and shape of XPath expressions, and only depends on certain characteristics of the XML stream, such as the data guide [11] or the graph schema [1,5]. These are small in many applications. Our theoretical results thus validate the use of a lazy DFA for XML stream processing. We verify these results experimentally, measuring the number of states in the lazy DFA for several synthetic and real data sets. We also confirm experimentally the performance of the lazy DFA, and find that a lazy DFA obtains constant throughput, independent of the number of XPath expressions.

The techniques described here are part of an open-source software package[1].

**Paper Organization.** We begin with an overview in Sec. 2 of the architecture in which the XPath expressions are used. We describe in detail processing with a DFA in Sec. 3, then discuss its construction in Sec. 4 and analyze its size, both theoretically and experimentally. Throughput experiments are discussed in Sec. 5. We discuss implementation issues in Sec. 6, and related work in Sec 7. Finally, we conclude in Sec. 8.

## 2 Overview

### 2.1 The Event-Based Processing Model

We start by describing the architecture of an XML stream processing system [4], to illustrate the context in which XPath expressions are used. The user specifies several correlated XPath expressions arranged in a tree, called the *query tree*. An input XML stream is first parsed by a SAX parser that generates a stream of *SAX events* (Fig. 1); this is input to the query processor that evaluates the XPath expressions and generates a stream of *application events*. The application is notified of these events, and usually takes some action such as forwarding the

[1] Described in [4] and available at `xmltk.sourceforge.net`.

packet, notifying a client, or computing some values. An optional Stream Index (called SIX) may accompany the XML stream to speed up processing [4]: we do not discuss the index here.

The query tree, $Q$, has nodes labeled with variables and the edges with linear XPath expressions, $P$, given by the following grammar:

$$P ::= /N \mid //N \mid PP \qquad N ::= E \mid A \mid text(S) \mid * \tag{1}$$

Here $E, A$, and $S$ are an element label, an attribute label, and a string constant respectively, and $*$ is the wild card. The function `text(S)` matches a text node whose value is the string `S`. While filters, also called predicates, are not explicitly allowed, we show below that they can be expressed. There is a distinguished variable, $\$R$, which is always bound to the root. We leave out from our presentation some system level details, for example the fact that the application may specify under which application events it wants to receive the SAX events. We refer the reader to [4] for system level details.

*Example 1.* The following is a query tree (tags taken from [19]):

```
$D IN $R/datasets/dataset    $H  IN $D/history
$T IN $D/title               $TH IN $D/tableHead
$N IN $D//tableHead//*       $F  IN $TH/field
$V IN $N/text("Galaxy")
```

Fig. 2 shows this query tree graphically. Fig. 3 shows the result of evaluating this query tree on an XML input stream: the first column shows the XML stream, the second shows the SAX events generated by the parser, and the last column shows the application events.

**Filters.** Currently our query trees do not support XPath expressions with filters (a.k.a. predicates). One can easily implement filters over query trees in a naive way, as we illustrate here on the following XPath expression:

```
$X IN $R/catalog/product[@category="tools"][sales/@price > 200]/quantity
```

First decompose it into several XPath expression, and construct the query tree $Q$ in Fig. 4. Next we use our query tree processor, and add the following actions. We declare two boolean variables, `b1, b2`. On a `$Z` event, set `b1` to `true`; on a `$U` event test the following text value and, if it is $> 200$, then set `b2` to `true`. At the end of a `$Y` event check whether `b1=b2=true`. This clearly implements the two filters in our example. Such a method can be applied to arbitrary filters and predicates, with appropriate bookkeeping, but clearly throughput will decrease with the number of filters in the query tree. Approaches along these lines are discussed in [3,6,9]. More advanced methods for handling filters include event detection techniques [20] or pushdown automata [21].

**The Event-based Processing Problem.** The problem that we address is: given a query tree $Q$, preprocesses it, then evaluate it on an incoming XML stream. The goal is to maximize the throughput at which we can process the XML stream. A special case of a query tree, $Q$, is one

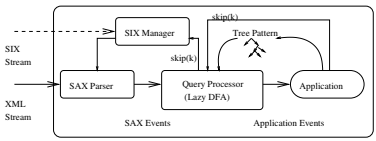



**Fig. 1.** System's Architecture

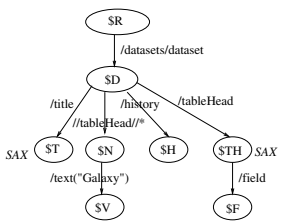



**Fig. 2.** A Query Tree

| XML Stream | Parser SAX Events | Variable Events |
|---|---|---|
| <datasets> | start(datasets) | start($R) |
| <dataset> | start(dataset) | start($D) |
| <history> | start(history) | start($H) |
| <date> | start(date) | |
| 10/10/59 | text("10/10/59") | |
| </date> | end(date) | |
| </history> | end(history) | end($H) |
| <title> | start(title) | start($T) |
| | | |
| <subtitle> | start(subtitle) | |
| $study | text(Study) | |
| </subtitle> | end(subtitle) | |
| </title> | end(title) | |
| | | end($T) |
| ... | | |
| </dataset> | end(dataset) | end($D) |
| ... | ... | |
| </datasets> | end(datasets) | end($R) |

**Fig. 3.** Events generated by a Query Tree

in which every node is either the root or a leaf node, i.e. has the form: $\$X_1$ in $\$R/e_1, \$X_2$ in $\$R/e_2, \ldots, \$X_p$ in $\$R/e_p$ (each $e_i$ may start with // instead of /): we call $Q$ a *query set*, or simply a *set*. Each query tree $Q$ can be rewritten into an equivalent query set $Q'$, as illustrated in Fig. 4.

```
Q:                                    Q':
$Y IN $R/catalog/product              $Y IN $R/catalog/product
$Z IN $Y/@category/text("tools")      $Z IN $R/catalog/product/@category/text("tools")
$U IN $Y/sales/@price                 $U IN $R/catalog/product/sales/@price
$X IN $Y/quantity                     $X IN $R/catalog/product/quantity
```

**Fig. 4.** A query tree $Q$ and an equivalent query set $Q'$.

## 3 Processing with DFAs

### 3.1 Background on DFAs

Our approach is to convert a query tree into a Deterministic Finite Automaton (DFA). Recall that the query tree may be a very large collection of XPath expressions: we convert *all* of them into a *single* DFA. This is done in two steps: convert the query tree into a Nondeterministic Finite Automaton (NFA), then convert the NFA to a DFA. We review here briefly the basic techniques for both steps and refer the reader to a textbook for more details, e.g. [14]. Our running example will be the query tree $P$ shown in Fig. 5(a). The NFA, denoted $A_n$, is illustrated in Fig. 5(b). Transitions labeled $*$ correspond to $*$ or // in $P$; there

is one initial state; there is one terminal state for each variable (`$X`, `$Y`, ... ); and there are $\varepsilon$-transitions [2]. It is straightforward to generalize this to any query tree. The number of states in $A_n$ is proportional to the size of $P$.

Let $\Sigma$ denote the set of all tags, attributes, and text constants occurring in the query tree $P$, plus a special symbol $\omega$ representing any other symbol that could be matched by $*$ or //. For $w \in \Sigma^*$ let $A_n(w)$ denote the set of states in $A_n$ reachable on input $w$. In our example we have $\Sigma = \{a, b, d, \omega\}$, and $A_n(\varepsilon) = \{1\}$, $A_n(ab) = \{3, 4, 7\}$, $A_n(a\omega) = \{3, 4\}$, $A_n(b) = \emptyset$.

The DFA for $P$, $A_d$, has the following set of states:

$$states(A_d) = \{A_n(w) \mid w \in \Sigma^*\} \tag{2}$$

For our running example $A_d$ is illustrated[3] in Fig. 5 (c). Each state has unique transitions, and one optional `[other]` transition, denoting any symbol in $\Sigma$ *except* the explicit transitions at that state: this is different from $*$ in $A_n$ which denotes *any* symbol. For example `[other]` at state $\{3, 4, 8, 9\}$ denotes either $a$ or $\omega$, while `[other]` at state $\{2, 3, 6\}$ denotes $a, d$, or $\omega$. Terminal states may be labeled now with more than one variable, e.g. $\{3, 4, 5, 8, 9\}$ is labeled `$Y` and `$Z`.

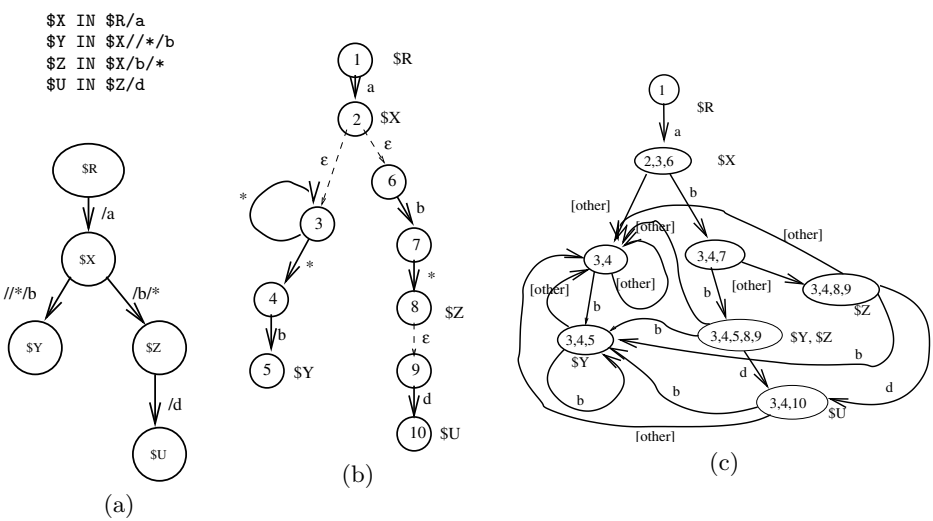



**Fig. 5.** (a) A query tree; (b) its NFA, $A_n$, and (c) its DFA, $A_d$.

[2] These are needed to separate the loops from the previous state. For example if we merge states 2, 3, and 6 into a single state then the $*$ loop (corresponding to //) would incorrectly apply to the right branch.

[3] Technically, the state $\emptyset$ is also part of the DFA, and behaves like a "failure" state, collecting all missing transitions. We do not illustrate it in our examples.

### 3.2 The DFA at Run Time

Processing an XML stream with a DFA is very efficient. We maintain a pointer to the current DFA state, and a stack of DFA states. SAX events are processed as follows. On a `start(element)` event we push the current state on the stack, and replace the state with the state reached by following the `element` transition[4]; on an `end(element)` we pop a state from the stack and set it as the current state. Attributes and `text(string)` are handled similarly. No memory management is needed at run time[5]. Thus, each SAX event is processed in $O(1)$ time, and we can guarantee the throughput, independent of the number of XPath expressions. The main issue is the size of the DFA, which we discuss next.

## 4 Analyzing the Size of the DFA

For a general regular expression the size of the DFA may be exponential [14]. In our setting, however, the expressions are restricted to XPath expressions defined in Sec. 2.1, and general lower bounds do not apply automatically. We analyze and discuss here the size of the eager and lazy DFAs for such XPath expressions. We shall assume first that the XPath expressions have no text constants (`text(S)`) and, as a consequence, the alphabet $\Sigma$ is small, then discuss in Sec. 4.4 the impact of the constants on the number of states. As discussed at the end of Sec.2 we will restrict our analysis to query trees that are sets.

### 4.1 The Eager DFA

**Single XPath Expression.** A linear XPath expression has the form $P = p_0//p_1//\ldots//p_k$ where each $p_i$ is $N_1/N_2/\ldots/N_{n_i}$, $i = 0, \ldots, k$, and each $N_j$ is given by (1). We consider the following parameters:

$$\begin{array}{ll} k = \text{ number of //'s} & n_i = \text{length of } p_i,\ i = 0, \ldots, k \\ m = \max \text{ \# of } *\text{'s in each } p_i & n = \text{ length of } P,\ \sum_{i=0,k} n_i \\ s = \text{ alphabet size} = \mid \Sigma \mid & \end{array}$$

For example if $P = //a/*//a/*/b/a/*/a/b$, then $k = 2$ ($p_0 = \varepsilon$, $p_1 = a/*$, $p_2 = a/*/b/a/*/a/b$), $s = 3$ ($\Sigma = \{a, b, \omega\}$), $n = 9$ (node tests: $a, *, a, *, b, a, *, a, b$), and $m = 2$ (we have 2 $*$'s in $p_2$). The following theorem gives an upper bound on the number of states in the DFA, and is, technically, the hardest result in the paper. The proof is in [12].

**Theorem 1.** *Given a linear XPath expression $P$, define prefix$(P) = n_0$ and suffix$(P) = k + k(n - n_0)s^m$. Then the eager DFA for $P$ has at most prefix$(P)$ + suffix$(P)$ states. In particular, when $k = 0$ the DFA has at most $n$ states, and when $k > 0$ the DFA has at most $k + kns^m$ states.*

[4] The state's transitions are stored in a hash table.

[5] The stack is a static array, currently set to 1024: this represents the maximum XML depth that we can handle.

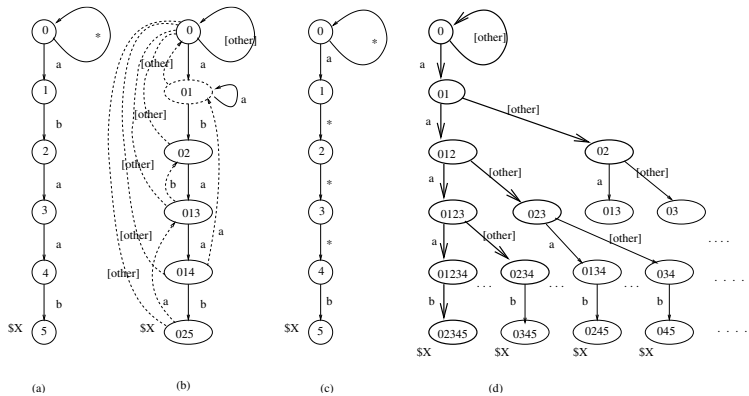



**Fig. 6.** The NFA (a) and the DFA (b) for `//a/b/a/a/b`. The NFA (c) and the DFA (with back edges removed) (d) for `//a/*/*/*/b`: here the eager DFA has $2^5 = 32$ states, while the lazy DFA, assuming the DTD `<!ELEMENT a (a*|b)>`, has at most 9 states.

We first illustrate the theorem in the case where there are no wild-cards ($m = 0$); then there are at most $k+kn$ states in the DFA. For example, if $p = //a/b/a/a/b$, then $k = 1, n = 5$: the NFA and DFA shown in Fig. 6 (a) and (b), and indeed the latter has 6 states. This generalizes to $//N_1/N_2/\ldots/N_n$: the DFA has only $n + 1$ states, and is an isomorphic copy of the NFA plus some back transitions: this corresponds to Knuth-Morris-Pratt's string matching algorithm [8].

When there are wild cards ($m > 0$), the theorem gives an exponential upper bound. There is a corresponding exponential lower bound, illustrated in Fig. 6 (c), (d), showing that the DFA for $p = //a/*/*/*/b$, has $2^5$ states. It is easy to generalize this example and see that the DFA for $//a/*/\ldots/*/b$ has $2^{m+2}$ states[6], where $m$ is the number of $*$'s.

Thus, the theorem shows that the only thing that can lead to an exponential growth of the DFA is the maximum number of $*$'s between any two consecutive //'s. One expects this number to be small in most practical applications; arguably users write expressions like `/catalog//product//color` rather than `/catalog//product/*/*/*/*/*/*/*/*/*/color`. Some implementations of XQuery already translate a *single* linear XPath expression into DFAs [15].

**Multiple XPath Expressions.** For sets of XPath expressions, the DFA also grows exponentially with the number expressions containing //. We illustrate first, then state the lower and upper bounds.

*Example 2.* Consider four XPath expressions:

```
$X1 IN $R//book//figure      $X2 IN $R//table//figure
$X3 IN $R//chapter//figure   $X4 IN $R//note//figure
```

The eager DFA needs to remember what subset of tags of

[6] The theorem gives the upper bound: $1 + (m + 2)3^m$.

{book, table, chapter, note} it has seen, resulting in at least $2^4$ states. We generalize this below.

**Proposition 1.** *Consider p XPath expressions:* $X_1 IN $R//a_1//b ... $X_p IN $R//a_p//b *where* $a_1, \ldots, a_p, b$ *are distinct tags. Then the DFA has at least* $2^p$ *states.*[7]

**Theorem 2.** *Let Q be a set of XPath expressions. Then the number of states in the eager DFA for Q is at most:* $\sum_{P \in Q}(prefix(P)) + \prod_{P \in Q}(1 + suffix(P))$ *In particular, if* $A, B$ *are constants s.t.* $\forall P \in Q$, $prefix(P) \leq A$ *and* $suffix(P) \leq B$, *then the number of states in the eager DFA is* $\leq p \times A + B^{p'}$, *where* $p'$ *is the number of XPath expressions* $P \in Q$ *that contain //.*

Recall that *suffix*(*P*) already contains an exponent, which we argued is small in practice. The theorem shows that the extra exponent added by having multiple XPath expressions is precisely the number of expressions with //'s. This result should be contrasted with Aho and Corasick's dictionary matching problem [2, 22]. There we are given a dictionary consisting of $p$ words, $\{w_1, \ldots, w_p\}$, and have to compute the DFA for the set $Q = \{//w_1, \ldots, //w_p\}$. Hence, this is a special case where each XPath expression has a single, leading //, and has no $*$. The main result in the dictionary matching problem is that the number of DFA states is linear in the total size of $Q$. Theorem 2 is weaker in this special case, since it counts each expression with a // toward the exponent. The theorem could be strengthened to include in the exponent only XPath expressions with at least two //'s, thus technically generalizing Aho and Corasick's result. However, XPath expressions with two or more occurrences of // *must* be added to the exponent, as Proposition 1 shows. We chose not to strengthen Theorem 2 since it would complicate both the statement and proof, with little practical significance.

Sets of XPath expressions like the ones we saw in Example 2 are common in practice, and rule out the eager DFA, except in trivial cases. The solution is to construct the DFA lazily, which we discuss next.

### 4.2 The Lazy DFA

The *lazy DFA* is constructed at run-time, on demand. Initially it has a single state (the initial state), and whenever we attempt to make a transition into a missing state we compute it, and update the transition. The hope is that only a small set of the DFA states needs to be computed.

This idea has been used before in text processing, but it has never been applied to such large number of expressions as required in our applications (e.g. 100,000): a careful analysis of the size of the lazy DFA is needed to justify its feasibility. We prove two results, giving upper bounds on the number of states

[7] Although this requires $p$ distinct tags, the result can be shown with only 2 distinct tags, and XPath expressions of depths $n = O(\log p)$, using standard techniques.

in the lazy DFA, that are specific to XML data, and that exploit either the schema, or the data guide. We stress, however, that neither the schema nor the data guide need to be known in order to use the lazy DFA, and only serve for the theoretical results.

Formally, let $A_l$ be the lazy DFA. Its states are described by the following equation which should be compared to Eq.(2):

$$states(A_l) = \{A_n(w) \mid w \in \mathcal{L}_{data}\} \tag{3}$$

Here $\mathcal{L}_{data}$ is the set of all root-to-leaf sequences of tags in the input XML streams. Assuming that the XML stream conforms to a schema (or DTD), denote $\mathcal{L}_{schema}$ all root-to-leaf sequences allowed by the schema: we have $\mathcal{L}_{data} \subseteq \mathcal{L}_{schema} \subseteq \Sigma^*$.

We use *graph schema* [1,5] to formalize our notion of schema, where nodes are labeled with tags and edges denote inclusion relationships. Define a simple cycle, $c$, in a graph schema to be a set of nodes $c = \{x_0, x_1, \ldots, x_{n-1}\}$ which can be ordered s.t. for every $i = 0, \ldots, n-1$, there exists an edge from $x_i$ to $x_{i+1 \mod n}$. We say that a graph schema is *simple*, if for any two cycles $c \neq c'$, we have $c \cap c' = \emptyset$.

We illustrate with the DTD in Fig. 7, which also shows its graph schema [1]. This DTD is simple, because the only cycles in its graph schema (shown in Fig. 7 (a)) are self-loops. All non-recursive DTDs are simple. For a simple graph schema we denote $d$ the maximum number of cycles that a simple paths can intersect (hence $d = 0$ for non-recursive schemes), and $D$ the total number of nonempty, simple paths: $D$ can be thought of as the number of nodes in the unfolding[8]. In our example $d = 2$, $D = 13$, and the unfolded graph schema is shown in Fig. 7 (b). For a query set $Q$, denote $n$ its depth, i.e. the maximum number of symbols in any $P \in Q$ (i.e. the maximum $n$, as in Sec. 4.1). We prove the following result in [12]:

**Theorem 3.** *Consider a simple graph schema with $d, D$, defined as above, and let $Q$ be a set of XPath expressions of maximum depth $n$. Then the lazy DFA has at most $1 + D \times (1 + n)^d$ states.*

The result is surprising, because the number of states does not depend on the number of XPath expressions, only on their depths. In Example 2 the depth is $n = 2$: for the DTD above, the theorem guarantees at most $1 + 13 \times 3^2 = 118$ states in the lazy DFA. In practice, the depth is larger: for $n = 10$, the theorem guarantees $\leq 1574$ states, even if the number of XPath expressions increases to, say, 100,000. By contrast, the eager DFA has $\geq 2^{100000}$ states (see Prop. 1). Fig. 6 (d) shows another example: of the $2^5$ states in the eager DFA only 9 are expanded in the lazy DFA.

[8] The constant $D$ may, in theory, be exponential in the size of the schema because of the unfolding, but in practice the shared tags typically occur at the bottom of the DTD structure (see [23]), hence $D$ is only modestly larger than the number of tags in the DTD.

Theorem 3 has many applications. First for *non-recursive* DTDs ($d = 0$) the lazy DFA has at most $1 + D$ states[9]. Second, in *data-oriented* XML instances, recursion is often restricted to hierarchies, e.g. departments within departments, parts within parts. Hence, their DTD is simple, and $d$ is usually small. Finally, the theorem also covers applications that handle documents from *multiple* DTDs (e.g. in XML routing): here $D$ is the sum over all DTDs, while $d$ is the maximum over all DTDs.

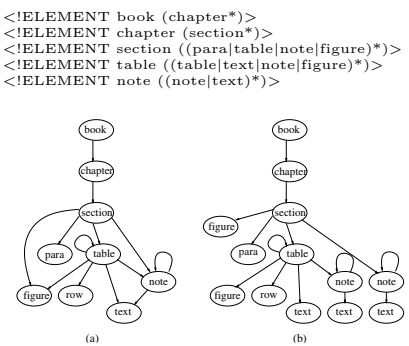



**Fig. 7.** A graph schema for a DTD (a) and its unfolding (b).

| DTD Source | DTD Names | (DTD Statistics) | | Data size MB |
|---|---|---|---|---|
| | | No. elms | Simple ? | |
| [synthetic] | simple.dtd | 12 | Yes | - |
| www.wapforum.org | prov.dtd | 3 | Yes | - |
| www.ebxml.org | ebBPSS.dtd | 29 | Yes | - |
| pir.georgetown.edu | protein.dtd | 66 | Yes | 684 |
| xml.gsfc.nasa.gov | nasa.dtd | 117 | No | 24 |
| UPenn Treebank | treebank.dtd | 249 | No | 56 |

**Fig. 8.** Sources of data used in experiments. Only three real data sets were available.

The theorem does not apply, however, to *document-oriented* XML data. These have non-simple DTDs : for example a `table` may contain a `table` or a `footnote`, and a `footnote` may also contain a `table` or a `footnote` (hence, both {`table`} and {`table`, `footnote`} are cycles, and they share a node). For such cases we give an upper bound on the size of the lazy DFA in terms of *Data Guides* [11]. The data guide is a special case of a graph schema, with $d = 0$, hence Theorem 3 gives:

**Corollary 1.** *Let $G$ be the number of nodes in the data guide of an XML stream. Then, for any set $Q$ of XPath expressions the lazy DFA for $Q$ on that XML stream has at most $1 + G$ states.*

An empirical observation is that real XML data tends to have small data guides, regardless of its DTD. For example users occasionally place a `footnote` within a `table`, or vice versa, but do not nest elements in all possible ways allowed by the schema. All XML data instances described in [16] have very small data guides, except for Treebank [17], where the data guide has $G = 340,000$ nodes.

**Using the Schema or DTD.** If a Schema or DTD is available, it is possible to specialize the XPath expressions and remove all $*$'s and //'s, and replace

[9] This also follows directly from (3) since in this case $\mathcal{L}_{schema}$ is finite and has $1 + D$ elements: one for $w = \varepsilon$, and one for each non-empty, simple paths.

them with general Kleene closures: this is called *query pruning* in [10]. For example for the schema in Fig. 7 (a), the expression `//table//figure` is pruned to `/book/chapter/section/(table)+/figure`. This offers no advantage to computing the DFA lazily, and should be treated orthogonally. Pruning may increase the number of states in the DFA by up to a factor of $D$: for example, the lazy (and eager) DFA for `//*` has only one state, but if we first prune it with respect to a graph schema with $D$ nodes, the DFA has $D$ states.

**Size of NFA Tables.** A major component of the space used by the lazy DFA are the sets of NFA states that need to be kept at each DFA state. We call these sets *NFA tables*. The following proposition is straightforward, and ensures that the NFA tables do not increase exponentially:

**Proposition 2.** *Let $Q$ be a set of $p$ XPath expressions, of maximum depths $n$. Then the size of each NFA table in the DFA for $Q$ is at most $n \times p$.*

Despite the apparent positive result, the sets of NFA states are responsible for most of the space in the lazy DFA, and we discuss them in Sec. 6.

### 4.3 Validation of the Size of the Lazy DFA

We ran experiments measuring the size of the lazy DFA for XML data for several publicly available DTDs, and one synthetic DTD. We generated synthetic data for these DTDs[10]. For three of the DTDs we also had access to real XML instances. The DTDs and the available XML instances are summarized in Fig. 8: four DTDs are simple, two are not; `protein.dtd` is non-recursive. We generated three sets of queries of depth $n = 20$, with 1,000, 10,000, and 100,000 XPath expressions[11], with `5%` probabilities for both the $*$ and the //.

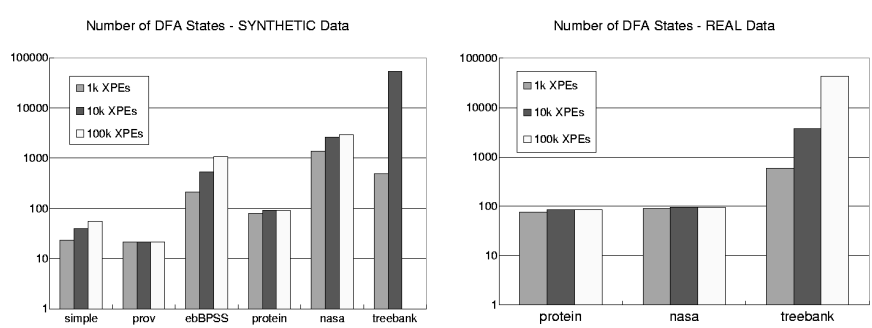



**Fig. 9.** Size of the lazy DFA for (left) synthetic data, and (right) real data. 1k means 1000 XPath expressions. For 100k XPath expressions for the `treebank` DTD with synthetic data we ran out of memory.

[10] Using `http://www.alphaworks.ibm.com/tech/xmlgenerator`.
[11] We used the generator described in [9].

Fig. 9(a) shows the number of states in the lazy DFA for the *synthetic* data. The first four DTDs are simple, or non-recursive, hence Theorem 3 applies. They had significantly less states than the upper bound in the theorem; e.g. `ebBPSS.dtd` has 1058 states, while the upper bound is 12,790 ($D = 29$, $d = 2$, $n = 20$). The last two DTDs were not simple, and neither Theorem 3 nor Corollary 1 applies (since synthetic data has large data guides). In one case (Treebank, 100,000 expressions) we ran out of memory.

Fig. 9(b) shows the number of states in the lazy DFA for *real* data. This is much lower than for synthetic data, because real data has small dataguides, and Corollary 1 applies; by contrast, the dataguide for synthetic data may be as large as the data itself. The `nasa.dtd` had a dataguide with 95 nodes, less than the number of tags in the DTD (117) because not all the tags occurred in the data. As a consequence, the lazy DFA had at most 95 states. Treebank has a data guide with 340,000 nodes; the largest lazy DFA here had only 44,000 states.

We also measured experimentally the average size of the NFA tables in each DFA state and found it to be around $p/10$, where $p$ is the number of XPath expressions (see [12] ). Proposition 2 also gives an upper bound $O(p)$, but the constant measured in the experiments is much lower than that in the Theorem. These tables use most of the memory space and we address them in Sec. 6. Finally, we measured the average size of the transition tables per DFA state, and found it to be small (less than 40).

### 4.4 Constant Values

Finally, we comment on the impact of constant values on the number of states in the DFA. Each linear XPath expression can now end in a `text(S)` predicate, see Eq.(1). For a given set of XPath expressions, $Q$, let $\Sigma$ denote the set of all symbols in $Q$, including those of the form `text(S)`. Let $\Sigma = \Sigma_t \cup \Sigma_s$, where $\Sigma_t$ contains all element and attribute labels and $\omega$, while $\Sigma_s$ contains all symbols of the form `text(S)`. The NFA for $Q$ has a special, 2-tier structure: first an NFA over $\Sigma_t$, followed by some $\Sigma_s$-transitions into sink states, i.e. with no outgoing transitions. The corresponding DFA also has a two-tier structure: first the DFA for the $\Sigma_t$ part, denote it $A^t$, followed by $\Sigma_s$ transitions into sink states. All our previous upper bounds on the size of the lazy DFA apply to $A^t$. We now have to count the additional sink states reached by `text(S)` transitions. For that, let $\Sigma_s = \{\texttt{text}(\texttt{S}_1), \ldots, \texttt{text}(\texttt{S}_q)\}$, and let $Q_i$, $i = 1, \ldots, q$, be the set of XPath expressions in $Q$ that end in $\texttt{text}(\texttt{S}_i)$; we assume w.l.o.g. that every XPath expression in $Q$ ends in some predicate in $\Sigma_s$, hence $Q = Q_1 \cup \ldots \cup Q_q$. Denote $A_i$ the DFA for $Q_i$, and $A_i^t$ its $\Sigma_t$-part. Let $s_i$ be the number of states in $A_i^t$, $i = 1, \ldots, q$. All the previous upper bounds, in Theorem 1, Theorem 3, and Corollary 1 apply to each $s_i$. We prove the following in [12].

**Theorem 4.** *Given a set of XPath expressions $Q$, containing $q$ distinct constant values of the form `text(S)`, the additional number of sink states in the lazy DFA due to the constant values is at most $\sum_{i=1,q} s_i$.*

## 5 Experiments

This section validates the throughput achieved by lazy DFAs in stream XML processing. Our execution environment consists of a dual 750MHz SPARC V9 with 2048MB memory, running SunOS 5.8. Our compiler is gcc version 2.95.2, without any optimization options.

We used the NASA XML dataset [19] and concatenated all the XML documents into one single file, which is about 25MB. We generated sets of 1k (= 1000), 10k, 100k, and 1000k XPath expression using the XPath generator from [9], and varied the probability of $*$ and // to 0.1%, 1%, 10%, and 50% respectively. We report the throughput as a function of each parameter, while keeping the other two constant. For calibration and comparison we also report the throughput for parsing the XML stream, and the throughput of XFilter [3], which we re-implemented, without list balancing.

Figure 10 shows our results. In (a) we show the throughput as a function of the number of XPath expressions. The most important observation is that in the stable state (after processing the first 5-10MB of data) the throughput was constant, about 5.4MB/s. Notice that this is about half the parser's throughput, which was about 10MB/s; of course, the XML stream needs to be parsed, hence 10MB/s should be seen as an upper bound on our platform. We observed in several other experiments with other datasets (not shown here) that the throughput is constant, i.e. independent on the number of XPath expressions. By contrast, the throughput of XFilter decreased linearly with the number of XPath expressions. The lazy DFA is about 50 times faster than XFilter on the smallest dataset, and about 10,000 times faster than XFilter on the largest dataset. Figure 10 (b) and (c) show the throughput as a function of the probability of $*$, and of the probability of // respectively.

The first 5MB-10MB of data in Fig. 10 represent the *warm-up phase*, when most of the states in the lazy DFA are constructed. The length of the warm-up phase depends on the size of the lazy DFA that is eventually generated. For the data in our experiments, the lazy DFA had the same number of states for 1k, 10k, 100k, and 1000k (91, 95, 95, and 95 respectively). However, the size of the NFA tables grows linearly with the number of XPath expressions, which explains the longer tail: even if few states remain to be constructed, they slow down processing. In our throughput experiments with other datasets we observed that the lengths of the warm-up phase is correlated to the number of states in the lazy DFA.

## 6 Implementation Issues

**Implementing the NFA Tables.** In the lazy DFA we need to keep the set of NFA states at each DFA state: we call this set an *NFA table*. As shown in Prop. 2 the size of an NFA table is linear in the number of XPath expressions $p$, and about $p/10$ in our experiments. Constructing and manipulating these tables during the warm-up phase is a significant overhead, both in space and in time.

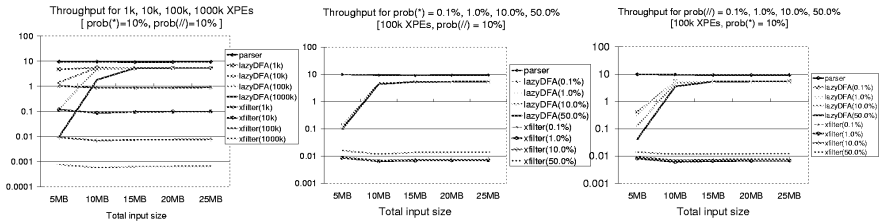



**Fig. 10.** Experiments illustrating the throughput of the DFA v.s. XFilter [3], as a function of the amount of XML data consumed. (left) varying number of XPath expressions (1k = 1000). (middle) varying probability of $*$. (right) varying probability of //.

We considered many alternative implementations for the NFA tables. There are three operations done on these sets: create, insert, and compare. For example a complex data set might have 10,000 DFA states, each containing a table of 30,000 NFA states and 50 transitions. Then, during warm-up phase we need to *create* $50 \times 10,000 = 500,000$ new sets; *insert* 30,000 NFA states in each set; and *compare*, on average, $500,000 \times 10,000/2$ pairs of sets, of which only 490,000 comparisons return `true`, the others return `false`. We found that implementing sets as *sorted arrays* of pointers offered the best overall performance. An insertion takes $O(1)$ time, because we insert at the end, and sort the array when we finish all insertions. We compute a hash value (signature) for each array, thus comparisons with negative answers take $O(1)$ in virtually all cases.

**Optimizing Space.** To save space, it is possible to delete some of the sets of NFA tables, and recompute them if needed: this may slow down the warm-up phase, but will not affect the stable state. It suffices to maintain in each DFA state a pointer to its predecessor state (from which it was generated). When the NFA table is needed, but has been deleted (a *miss*), we re-compute it from the predecessor's set; if that is not available, then we go to *its* predecessor, eventually reaching the initial DFA state for which we always keep the NFA table.

**Updates.** Both *online* and *offline* updates to the set of XPath expressions are possible. In the *online* update, when a new XPath expression is inserted we construct its NFA, then create a new lazy DFA for the union of this NFA and the old lazy DFA. The new lazy DFA is very efficient to build (i.e. its warm-up is fast) because it only combines two automata, of which one is deterministic and the other is very small. When another XPath expression is inserted, then we create a new lazy DFA. This results in a hierarchy of lazy DFAs, each constructed from one NFA and another lazy DFA. A state expansion at the top of the hierarchy may cascade a sequence of expansions throughout the hierarchy. Online deletions are implemented as invalidations: reclaiming the memory used by the deleted XPath expressions requires garbage-collection or reference count. *Offline* updates can be done by a separate (offline) system, different from the production system. Copy the current lazy DFA, $A_l$, on the offline system, and also copy there the new

query tree, $P$, reflecting all updates (insertions, deletions, etc). Then construct the eager DFA, $A_d$, for $P$, but only expand states that have a corresponding state in $A_l$, by maintaining a one-to-one correspondence from $A_d$ to $A_l$ and only expanding a state when this correspondence can be extended to the new state. When completed, $A_d$ is moved to the online system and processing resumes normally. The idea is that $A_d$ will be no larger than $A_l$ and, if there are only few updates, then $A_d$ will be approximately the same as $A_l$, meaning that the warm-up cost for $A_d$ is minimal.

## 7 Related Work

Two techniques for processing XPath expressions have been proposed. XFilter [3], its successor YFilter [9] and XTrie [6] evaluate large numbers of XPath expressions with what is essentially a highly optimized NFA. There is a space guarantee which is proportional to the total size of all XPath expressions. An optimization in XFilter, called list balancing can improve the throughput by factors of 2 to 4. XTrie identifies common strings in the XPath expressions and organizes them in a Trie. At run-time an additional data structure is maintained in order to keep track of the interaction between the substrings. The throughput in XTrie is about 2 to 4 times higher than that in XFilter with list balancing.

In [20] the authors describe a technique for event detection. Events are sets of atomic events, and they trigger queries defined by other sets of events. The technique here is also a variation on the Trie data structure. This is an efficient event detection method that can be combined with lazy DFAs in order to process XPath expressions with filters.

Reference [15] describes a general-purpose XML query processor that, at the lowest level, uses an event based processing model, and show how such a model can be integrated with a highly optimized XML query processor. We were influenced by [15] in designing our stream processing model. Query processors like [15] can benefit from an efficient low-level stream processor. Specializing regular expressions w.r.t. schemes is described in [10,18].

## 8 Conclusion

The challenge in fast XML stream processing with DFAs is that memory requirements have exponential bounds in the worst case. We proved useful theoretical bounds and validated them experimentally, showing that memory usage is manageable for *lazy* DFAs. We also validated lazy DFAs on stream XML data and found that they outperform previous techniques by factors of up to 10,000.

**Acknowledgments.** We thank Peter Buneman, AnHai Doan, Ashish Gupta, Zack Ives, and Arnaud Sahuguet for their comments on earlier versions of this paper. Suciu was partially supported by the NSF CAREER Grant 0092955, a gift from Microsoft, and an Alfred P. Sloan Research Fellowship.

## References


1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
2. A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.
3. M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination. In *Proceedings of VLDB*, pages 53–64, Cairo, Egypt, September 2000.
4. I. Avila-Campillo, T. J. Green, A. Gupta, M. Onizuka, D. Raven, and D. Suciu. XMLTK: An XML toolkit for scalable XML stream processing. In *Proceedings of PLANX*, October 2002.
5. P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, pages 336–350, Delphi, Greece, 1997. Springer Verlag.
6. C. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the International Conference on Data Engineering*, 2002.
7. J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for internet databases. In *Proceedings of the ACM/SIGMOD Conference on Management of Data*, pages 379–390, 2000.
8. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
9. Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of xml documents. In *Proceedings of the International Conference on Data Engineering*, San Jose, California, February 2002.
10. M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the International Conference on Data Engineering*, pages 14–23, 1998.
11. R. Goldman and J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of Very Large Data Bases*, pages 436–445, September 1997.
12. T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing xml streams with deterministic automata. Technical Report 02-10-03, University of Washington, 2002. Available from `www.cs.washington.edu/homes/suciu`.
13. D. G. Higgins, R. Fuchs, P. J. Stoehr, and G. N. Cameron. The EMBL data library. *Nucleic Acids Research*, 20:2071–2074, 1992.
14. J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.
15. Z. Ives, A. Halevy, and D. Weld. An XML query engine for network-bound data. Unpublished, 2001.
16. H. Liefke and D. Suciu. XMill: an efficent compressor for XML data. In *Proceedings of SIGMOD*, pages 153–164, Dallas, TX, 2000.
17. M. Marcus, B. Santorini, and M.A.Marcinkiewicz. Building a large annotated corpus of English: the Penn Treenbak. *Computational Linguistics*, 19, 1993.
18. J. McHugh and J. Widom. Query optimization for XML. In *Proceedings of VLDB*, pages 315–326, Edinburgh, UK, September 1999.
19. NASA's astronomical data center. ADC XML resource page. http://xml.gsfc.nasa.gov/.

20. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the web. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 437–448, Santa Barbara, 2001.
21. D. Olteanu, T. Kiesling, and F. Bry. An evaluation of regular path expressions with qualifiers against XML streams. In *Proc. the International Conference on Data Engineering*, 2003.
22. G. Rozenberg and A. Salomaa. *Handbook of Formal Languages*. Springer Verlag, 1997.
23. A. Sahuguet. Everything you ever wanted to know about dtds, but were afraid to ask. In D. Suciu and G. Vossen, editors, *Proceedings of WebDB*, pages 171–183. Sringer Verlag, 2000.
24. A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using XML. In *Proceedings of the 18th Symposium on Operating Systems Principles*, 2001.
25. J. Thierry-Mieg and R. Durbin. Syntactic Definitions for the ACEDB Data Base Manager. Technical Report MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge,CB2 2QH, UK, 1992.

# Deciding Termination of Query Evaluation in Transitive-Closure Logics for Constraint Databases

Floris Geerts*[1] and Bart Kuijpers[2]

[1] University of Helsinki
Helsinki Institute for Information Technology
PO Box 26 (Teollisuuskatu 23), Fin-00014, Helsinki, Finland
floris.geerts@cs.helsinki.fi
[2] University of Limburg
Dept. of Mathematics, Physics and Computer Science
Universitaire Campus, 3590 Diepenbeek, Belgium
bart.kuijpers@luc.ac.be

**Abstract.** We study extensions of first-order logic over the reals with different types of transitive-closure operators as query languages for constraint databases that can be described by Boolean combinations of polynomial inequalities over the reals. We are in particular interested in deciding the termination of the evaluation of queries expressible in these transitive-closure logics. It turns out that termination is undecidable in general. However, we show that the termination of the transitive closure of a continuous function graph in the two-dimensional plane, viewed as a binary relation over the reals, is decidable, and even expressible in first-order logic over the reals. Based on this result, we identify a particular transitive-closure logic for which termination of query evaluation is decidable and which is more expressive than first-order logic over the reals. Furthermore, we can define a guarded fragment in which exactly the terminating queries of this language are expressible.

## 1 Introduction

The framework of *constraint databases*, introduced in 1990 by Kanellakis, Kuper and Revesz [12] and by now well-studied [17], provides an elegant and powerful model for applications that deal with infinite sets of points in some real space $\mathbb{R}^n$, for instance spatial databases. In the setting of the constraint model, these infinite sets are finitely represented as Boolean combinations of polynomial equalities and inequalities over the reals. For example, the spatial database consisting of the set of points on the northern hemisphere together with the points on the equator of the unit sphere in the three-dimensional space $\mathbb{R}^3$ can be represented by the formula $x^2 + y^2 + z^2 = 1 \wedge z \geq 0$. The relational calculus augmented with polynomial constraints, FO for short, is the

* The research presented here was done while this author was at the University of Limburg.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 190–206, 2003.

standard first-order query language for constraint databases. The FO-sentence $(\exists r)(\forall x)(\forall y)(\forall z)(S(x,y,z) \rightarrow x^2 + y^2 + z^2 < r^2)$ expresses that the three-dimensional spatial relation $S$ is bounded.

Although many interesting properties can be expressed in FO, its most important deficiency is that its expressive power is rather limited. For instance, several practically relevant topological properties of spatial data, such as connectivity and reachability, are not expressible in FO and various people have proposed and studied extensions of FO with tractable recursion mechanisms to obtain more expressive languages [2,11,14,15,16]. In analogy with the classical graph connectivity query, which cannot be expressed in the standard relational calculus but which can be expressed in the relational calculus augmented with a transitive-closure operator, also extensions of FO with various *transitive-closure operators* have been proposed to obtain languages that are more expressive, in particular that allow the expression of connectivity and reachability queries and that are even computationally complete. Recently, the present authors introduced FO+TC and FO+TCS, two languages in which an operator is added to FO that allows the computation of the transitive closure of unparameterized sets in some $\mathbb{R}^{2k}$ [10]. In the latter language also FO-definable stop conditions are allowed to control the evaluation of the transitive-closure. Later on, Kreutzer has studied the language that we refer to as FO+KTC [15], which is an extension of FO with a transitive-closure operator that may be applied to parameterized sets and in which the evaluation of a transitive-closure expression may be controlled by the termination of particular paths in its computation rather than by the termination of the transitive closure of the complete set. The fragments of FO+TCS and FO+KTC, that does not use multiplication, are shown to be computationally complete on databases definable by linear constraints [10,15].

In all of these transitive-closure languages, we face the well-know fact that recursion involving arithmetic over an infinite domain, such as the reals with addition and multiplication in this setting, is not guaranteed to terminate. In this paper, we are interested in termination of query evaluation in these different transitive-closure logics and in particular in *deciding termination*. We show that the termination of the evaluation of a given query, expressed in any of these languages, on a given input database is undecidable as soon as the transitive closure of 4-ary relations is allowed. In fact, a known undecidable problem in *dynamical systems theory*, namely deciding *nilpotency* of functions from $\mathbb{R}^2$ to $\mathbb{R}^2$ [3,4], can be reduced to our decision problem. When the transitive-closure operator is restricted to work on binary relations, the matter is more complicated. We show the undecidability of termination for FO+TCS restricted to binary relations. However, both for FO+TC and FO+KTC restricted to binary relations, finding an algorithm for deciding termination would also solve some outstanding open problems in dynamical systems theory. Indeed, a decision procedure for FO+TC restricted to binary relations would solve the *nilpotency problem* for functions from $\mathbb{R}$ to $\mathbb{R}$ and a decision procedure for FO+KTC restricted to binary relations would solve the *point-to-fixed-point problem*. Both these problems are already open for some time [3,13].

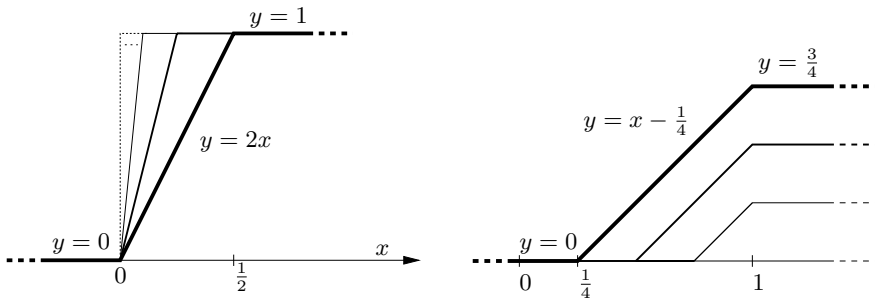



**Fig. 1.** On the left, a function graph (thick) with non-terminating transitive closure (thin). On the right, a function graph (thick) with terminating transitive closure (thin).

For FO+TC restricted to binary relations, we have obtained a positive decidability result, however. A basic problem in this context is deciding whether the transitive closure of a fixed subset of the two-dimensional plane, viewed as a binary relation over the reals, terminates. Even if these subsets are restricted to be the graphs of possibly discontinuous functions from $\mathbb{R}$ to $\mathbb{R}$, this problem is already puzzling dynamical system theorists for a number of years (it relates to the above mentioned point-to-fixed-point problem). However, when we restrict our attention to the transitive closure of *continuous function graphs*, we can show that the termination of the transitive closure of these figures is decidable. As an illustration of possible inputs for this decision problem, two continuous function graphs are given in Fig. 1. The one on the left has a non-terminating transitive closure, but the one on the right terminates after four iterations. Furthermore, we show that this decision procedure is expressible in FO. In the course of our proof, we also give a stronger version of Sharkovskiĭ's theorem [1] from dynamical systems theory for terminating continuous functions. We also extend another result in this area, namely, we show that nilpotency of continuous semi-algebraic functions is decidable and that this decision procedure is even expressible in FO. Previously, this result was only stated, without proof, for continuous piecewise affine functions [4].

Based on this decision result, we define a fragment of FO+TC in which the transitive-closure operator is restricted to work on graphs of continuous functions from $\mathbb{R}$ to $\mathbb{R}$. Termination of queries in this language is shown to be decidable. Furthermore, we define a *guarded* fragment of this transitive-closure logic in which only, and all, terminating queries can be formulated. We also show that this very restricted form of transitive closure yields a language that is strictly more expressive than FO.

This paper is organized as follows. In Section 2, we define constraint databases, the query language FO and several extensions with transitive-closure operators. In Section 3, we give general undecidability results. In Section 4, we give a procedure to decide termination of the transitive closure of continuous function graphs in the plane. In Section 5, we study the extension of FO with a transitive closure operator that is restricted to work on continuous function

graphs. In this section, we also describe a guarded fragment of this language and give expressiveness results. The paper concludes with some remarks.

## 2 Definitions and Preliminaries

In this section, we define constraint databases and their standard first-order query language FO. We also define existing extensions of this logic with different transitive-closure operators: FO+TC, FO+TCS and FO+KTC.

### 2.1 Constraint Databases and First-Order Logic over the Reals

Let $\mathbb{R}$ denote the set of the real numbers, and $\mathbb{R}^n$ the $n$-dimensional real space (for $n \geq 1$).

**Definition 1.** An $n$-dimensional *constraint database*[1] is a geometrical figure in $\mathbb{R}^n$ that can be defined as a Boolean combination (union, intersection and complement) of sets of the form $\{(x_1, \ldots, x_n) \mid p(x_1, \ldots, x_n) > 0\}$, where $p(x_1, \ldots, x_n)$ is a polynomial with integer coefficients in the real variables $x_1, \ldots, x_n$. □

We remark that in mathematical terminology, constraint databases are called *semi-algebraic sets* [5]. If a constraint database can be described by linear polynomials only, we refer to it as a *linear constraint database*.

In this paper, we will use FO, the relational calculus augmented with polynomial inequalities as a basic query language.

**Definition 2.** A formula in FO, over an $n$-dimensional input database, is a first-order logic formula, $\varphi(y_1, \ldots, y_m, S)$, built, using the logical connectives and quantification over real variables, from two kinds of atomic formulas, namely $S(x_1, \ldots, x_n)$ and $p(x_1, \ldots, x_k) > 0$, where $S$ is a $n$-ary relation name representing the input database and $p(x_1, \ldots, x_k)$ is a polynomial with integer coefficients in the real variables $x_1, \ldots, x_k$. □

In the expression $\varphi(y_1, \ldots, y_m, S)$, $y_1, \ldots, y_m$ denote the free variables. Variables in such formulas are assumed to range over $\mathbb{R}$. Tarski's quantifier-elimination procedure for first-order logic over the reals guarantees that FO expressions can be evaluated effectively on constraint database inputs and their result is a constraint database (in $\mathbb{R}^m$) that also can be described by means of polynomial constraints over the reals [6,21].

If $\varphi(y_1, \ldots, y_m, S)$ is an FO formula, $a_1, \ldots, a_m$ are reals, and $A$ is an $n$-dimensional constraint database, then we denote by $(a_1, \ldots, a_m, A) \models \varphi(y_1, \ldots, y_m, S)$ that $(a_1, \ldots, a_m, A)$ satisfies $\varphi$. We denote by $\varphi(A)$ the set $\{(a_1, \ldots, a_m) \mid (a_1, \ldots, a_m, A) \models \varphi(y_1, \ldots, y_m, S)\}$.

The fragment of FO in which multiplication is disallowed is called $\mathrm{FO}_{\mathrm{Lin}}$. This fragment is closed on the class of linear constraint databases [17].

[1] Spatial databases in the constraint model are usually defined as finite collections of such geometrical figures. We have chosen the simpler definition of a database as a single geometrical figure, but all results carry over to the more general setting.

### 2.2 Transitive-Closure Logics

We now define a number of extensions of FO (and of $\mathrm{FO}_{\mathrm{Lin}}$) with different types of transitive-closure operators. Recently, we introduced and studied the first two extensions, FO+TC and FO+TCS [9,10]. The latter extension, FO+KTC, is due to Kreutzer [15].

**Definition 3.** A formula in FO+TC is a formula built in the same way as an FO formula, but with the following extra formation rule: if $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a formula with $\boldsymbol{x}$ and $\boldsymbol{y}$ $k$-tuples of real variables, and with all free variables of $\psi$ among $\boldsymbol{x}$ and $\boldsymbol{y}$ and if $\boldsymbol{s}$, $\boldsymbol{t}$ are $k$-tuples of real variables, then

$$[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y})](\boldsymbol{s}, \boldsymbol{t}) \tag{1}$$

is also a formula which has as free variables those in $\boldsymbol{s}$ and $\boldsymbol{t}$. □

The semantics of a subformula of the above form (1) evaluated on a database $A$ is defined in the following operational manner: Start computing the following iterative sequence of $2k$-ary relations: $X_0 := \psi(A)$ and $X_{i+1} := X_i \cup \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{2k} \mid (\exists \boldsymbol{z})\,(X_i(\boldsymbol{x}, \boldsymbol{z}) \wedge X_0(\boldsymbol{z}, \boldsymbol{y}))\}$ and stop as soon as $X_i = X_{i+1}$. The semantics of $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y})](\boldsymbol{s}, \boldsymbol{t})$ is then defined as $(\boldsymbol{s}, \boldsymbol{t})$ belonging to the $2k$-ary relation $X_i$.

Since every step in the above algorithm, including the test for $X_i = X_{i+1}$, is expressible in FO, every step is effective and the only reason why the evaluation may not be effective is that the computation does *not terminate*. In that case the semantics of the formula (1) (and any other formula in which it occurs as subformula) is undefined.

As an example of an FO+TC formula over a 2-dimensional input database, we take $[\mathrm{TC}_{x;y}\, S(x, y)](s, t)$. This expression, when applied to a 2-dimensional figure, returns the transitive closure of this figure, viewed as a binary relation over $\mathbb{R}$. For illustrations of the evaluation of this formula, we refer to the examples in Fig. 1 in the Introduction.

The language $\mathrm{FO}_{\mathrm{Lin}}$+TC consists of all FO+TC formulas that do not use multiplication.

The following language, FO+TCS, is a modification of FO+TC that incorporates a construction to specify explicit termination conditions on transitive closure computations.

**Definition 4.** A formula in FO+TCS is built in the same way as an FO formula, but with the following extra formation rule: if $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a formula with $\boldsymbol{x}$ and $\boldsymbol{y}$ $k$-tuples of real variables; $\sigma$ is an FO sentence over the input database and a special $2k$-ary relation name $X$; and $\boldsymbol{s}$, $\boldsymbol{t}$ are $k$-tuples of real variables, then

$$[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y}) \mid \sigma](\boldsymbol{s}, \boldsymbol{t}) \tag{2}$$

is also a formula which has as free variables those in $\boldsymbol{s}$ and $\boldsymbol{t}$. We call $\sigma$ the *stop condition* of this formula. □

The semantics of a subformula of the above form (2) evaluated on databases $A$ is defined in the same manner as in the case without stop condition, but now we stop not only in case an $i$ is found such that $X_i = X_{i+1}$, but also when an $i$ is found such that $(A, X_i) \models \sigma$, whichever case occurs first. As above, we also consider the restriction $\mathrm{FO}_{\mathrm{Lin}}$+TCS. It was shown that $\mathrm{FO}_{\mathrm{Lin}}$+TCS is computationally complete, in the sense of Turing-complete on the polynomial constraint representation of databases (see Chap. 2 of [17]) on linear constraint databases [10].

Finally, we define FO+KTC. In finite model theory [8], transitive-closure logics, in general, allow the use of parameters. Also the language FO+KTC allows parameters in the transitive closure.

**Definition 5.** A formula in FO+KTC is a formula built in the same way as an FO formula, but with the following extra formation rule: if $\psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})$ is a formula with $\boldsymbol{x}$ and $\boldsymbol{y}$ $k$-tuples of real variables, $\boldsymbol{u}$ some further $\ell$-tuple of free variables, and where $\boldsymbol{s}$, $\boldsymbol{t}$ are $k$-tuples of real terms, then

$$[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})](\boldsymbol{s}, \boldsymbol{t}) \tag{3}$$

is also a formula which has as free variables those in $\boldsymbol{s}$, $\boldsymbol{t}$ and $\boldsymbol{u}$. □

Since the free variables in $\psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})$ are those in $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{u}$, here parameters are allowed in applications of the TC operator. The semantics of a subformula of the form (3), with $\boldsymbol{s} = (s_1, \ldots, s_k)$, evaluated on a database $A$ is defined in the following operational manner: Let $I$ be the set of indices $i$ for which $s_i$ is a constant. Then we start computing the following iterative sequence of $(2k + \ell)$-ary relations: $X_0 := \psi(A) \wedge \bigwedge_{i \in I}(s_i = x_i)$ and $X_{i+1} := X_i \cup \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) \in \mathbb{R}^{2k+\ell} \mid (\exists \boldsymbol{z})\,(X_i(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}) \wedge \psi(\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{u}))\}$ and stop as soon as $X_i = X_{i+1}$. The semantics of $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})](\boldsymbol{s}, \boldsymbol{t})$ is then defined as $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u})$ belonging to the $(2k+\ell)$-ary relation $X_i$.

We again also consider the fragment $\mathrm{FO}_{\mathrm{Lin}}$+KTC of this language. It was shown that $\mathrm{FO}_{\mathrm{Lin}}$+KTC is computationally complete on linear constraint databases [15].

For all of the transitive-closure logics that we have introduced in this section, we consider fragments in which the transitive-closure operator is restricted to work on relations of arity at most $2k$ and we denote this by adding $2k$ as a superscript to the name of the language. For example, in the language $\mathrm{FO+TCS}^4$, the transitive closure is restricted to binary and 4-ary relations.

## 3 Undecidability of the Termination of the Evaluation of Transitive-Closure Formulas

The decision problems that we consider in this section and the next take couples $(\varphi, A)$ as input, where $\varphi$ is an expression in the transitive-closure logic under consideration and $A$ is an input database, and the answer to the decision problem

is *yes* if the computation of the semantics of $\varphi$ on $A$ (as defined for that logic) terminates. We then say, for short, that $\varphi$ *terminates on* $A$.

We give the following general undecidability result concerning termination. In the proof and further on, the notion of nilpotency of a function will be used: a function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called *nilpotent* if there exists a natural number $k \geq 1$ such that for all $\boldsymbol{x} \in \mathbb{R}^n$, $f^k(\boldsymbol{x}) = (0, \ldots, 0)$.

**Theorem 1.** *It is undecidable whether a given formula in* FO+TC$^4$ *terminates on a given input database.*

Proof (sketch). We reduce deciding whether a piecewise affine function[2] $f : \mathbb{R}^2 \to \mathbb{R}^2$ is nilpotent to deciding whether the evaluation of a formula in FO+TC$^4$ terminates. So, assume that termination of formulas in FO+TC$^4$ is decidable. For a given piecewise affine function $f : \mathbb{R}^2 \to \mathbb{R}^2$, graph$(f)$, the graph of $f$, is a semi-algebraic subset of $\mathbb{R}^4$. We give a procedure to decide whether $f$ is nilpotent:

*Step 1.* Decide whether the FO+TC$^4$-query $[\mathrm{TC}_{x_1,x_2;y_1,y_2}\, S(x_1, x_2, y_1, y_2)](s_1, s_2, t_1, t_2)$ terminates on the input graph$(f)$; if the answer is *no*, then return *no*, else continue with Step 2.

*Step 2.* Compute $f^1(\mathbb{R}^2), f^2(\mathbb{R}^2), f^3(\mathbb{R}^2), \ldots$ and return *yes* if this ends with $\{(0,0)\}$, else return *no*.

This algorithm decides correctly whether $f$ is nilpotent, since for a nilpotent $f$, the evaluation of the transitive closure of graph$(f)$ will terminate, and the process in Step 2 is therefore also guaranteed to terminate. Since nilpotency of piecewise affine functions from $\mathbb{R}^2$ to $\mathbb{R}^2$ is known to be undecidable [4], this completes the proof. □

The following corollary follows from the previous theorem and the fact that FO+TC$^4$-formulas are in FO+KTC$^4$.

**Corollary 1.** *It is undecidable whether a given formula in* FO+KTC$^4$ *terminates on a given input database.* □

For transitive-closure logics with stop-condition, we even have undecidability for transitive closure restricted to binary relations.

**Theorem 2.** *It is undecidable whether a given formula in* FO+TCS$^2$ *terminates on a given input database.*

Proof (sketch). We prove this result by reducing the undecidability of a variant of Hilbert's 10th problem to it. This problem is deciding whether a polynomial $p(x_1, \ldots, x_{13})$ in 13 real variables and with integer coefficients has a solution in

[2] A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called *piecewise affine* if its graph is a linear semi-algebraic subset of $\mathbb{R}^n \times \mathbb{R}^n$.

the natural numbers [7,19]. For any such polynomial $p(x_1, \ldots, x_{13})$, let $\sigma_p$ be the FO-expressible stop-condition:

$$(\exists x_1) \cdots (\exists x_{13})(\bigwedge_{i=1}^{13} X(-1, x_i) \wedge p(x_1, \ldots, x_{13}) = 0).$$

Since, in consecutive iterations of the computation of the transitive closure of the graph of $y = x + 1$, $-1$ is mapped to $0, 1, 2, \ldots$, it is easy to see that $p(x_1, \ldots, x_{13})$ has an integer solution if and only if $[\mathrm{TC}_{x;y}\,(y = x + 1) \mid \sigma_p](s, t)$ terminates. Since the above mentioned Diophantine decision problem is undecidable, this completes the proof. □

The results in this section are complete for the languages FO+TC, FO+TCS and FO+KTC, apart from the cases FO+$\mathrm{TC}^2$ and FO+$\mathrm{KTC}^2$. The former case will be studied in the next sections. For the latter case, we remark that an open problem in dynamical systems theory, namely, the *point-to-fixed-point problem* reduces to it. This open problem is the decision problem that asks whether for a given algebraic number $x_0$ and a given piecewise affine function $f : \mathbb{R} \to \mathbb{R}$, the sequence $x_0, f(x_0), f^2(x_0), f^3(x_0), \ldots$ reaches a fixed point. Even for piecewise linear functions with two non-constant pieces this problem is open [3,13]. It is clear that this point-to-fixed-point problem can be expressed in FO+$\mathrm{KTC}^2$. So, we are left with the following unsolved problem.

**Open problem 1.** *Is it decidable whether a given formula in* FO+$\mathrm{KTC}^2$ *terminates on a given input database?* □

## 4 Deciding Termination for Continuous Function Graphs in the Plane

In this section, we study the termination of the transitive closure of a fixed semi-algebraic subset of the plane, viewed as a binary relation over $\mathbb{R}$. We say that a subset $A$ of $\mathbb{R}^2$ has a *terminating transitive closure*, if the query expressed by $[\mathrm{TC}_{x;y}\, S(x, y)](s, t)$ terminates on input $A$ using the semantics of FO+TC. In the previous section, we have shown that deciding nilpotency of functions can be reduced to deciding termination of the transitive closure of their function graphs. However, since it is not known whether nilpotency of (possible discontinuous) functions from $\mathbb{R}$ to $\mathbb{R}$ is decidable, we cannot use this reduction to obtain the undecidability in case of binary function graphs. We therefore have another unsolved problem:

**Open problem 2.** *Is it decidable whether a given formula in* FO+$\mathrm{TC}^2$ *terminates on a given input database?* □

There are obviously classes of functions for which deciding termination of their function graphs is trivial. An example is the class of the piecewise constant functions. In this section, we concentrate on a class that is non-trivial, namely

the class of the *continuous semi-algebraic*[3] *functions from* $\mathbb{R}$ *to* $\mathbb{R}$. The main purpose of this section is to prove the following theorem.

**Theorem 3.** *There is a decision procedure that on input a continuous semi-algebraic function* $f : \mathbb{R} \to \mathbb{R}$ *decides whether the transitive closure of* $\mathrm{graph}(f)$ *terminates. Furthermore, this decision procedure can be expressed by a formula in* FO *(over a 2-ary database that represents the graph of the input function).* □

Before we arrive at the proof of Theorem 3, we give a series of six technical lemma's. First, we introduce some terminology.

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function and let $x$ be a real number. We call the set $\{f^k(x) \mid k \geq 0\}$ the *orbit of* $x$ (with respect to $f$). A real number $x$ is said to be a *periodic point of* $f$ if $f^d(x) = x$ for some natural number $d \geq 1$. And we call the smallest such $d$ the *period of* $x$ (with respect to $f$). Let $Per(f)$ be the set of periodic points of $f$. If a real number $x$ is not a periodic point of $f$, but if $f^k(x)$ is periodic for some natural number $k \geq 1$, we call $x$ an *eventually periodic point of* $f$ and we call the smallest such number $k$ the *run-up of* $x$ (with respect to $f$). Finally, we call $f$ *terminating* if $\mathrm{graph}(f)$ has a terminating transitive closure.

The following lemma holds for arbitrary functions, not only for continuous ones. We also remark that Lemmas 1–4 hold for arbitrary functions, not only for semi-algebraic ones. We omit the proofs of Lemmas 1–4.

**Lemma 1.** *The function* $f : \mathbb{R} \to \mathbb{R}$ *is terminating if and only if there exist natural numbers* $k$ *and* $d$ *such that for each* $x \in \mathbb{R}$, $f^k(x)$ *is a periodic point of* $f$ *of period at most* $d$. □

**Lemma 2.** *Let* $f : \mathbb{R} \to \mathbb{R}$ *be a continuous function. If* $f$ *is terminating, then* $Per(f)$ *is a non-empty, closed and connected part of* $\mathbb{R}$. *In particular,* $Per(f) = f^k(\mathbb{R})$ *for some* $k \geq 1$. □

**Lemma 3.** *Let* $C$ *be a non-empty, closed and connected part of* $\mathbb{R}$. *If* $f : C \to C$ *is a continuous function and if every* $x \in C$ *is periodic for* $f$, *then* $f$ *or* $f^2$ *is the identity mapping on* $C$. □

**Lemma 4.** *For a continuous and terminating* $f : \mathbb{R} \to \mathbb{R}$, $Per(f) = \{x \in \mathbb{R} \mid f^2(x) = x\}$. □

Denote by $C_i$ the set of fixed points of $f^i$, i.e., the set of $x \in \mathbb{R}$ for which $f^i(x) = x$. From the previous lemmas it follows that for continuous and terminating $f$,

$$Per(f) = C_1 \cup C_2,$$

and that either $C_2 \setminus C_1$ is empty and $C_1$ is non-empty or $C_2 \setminus C_1$ is non-empty and $C_1$ is a singleton with the points of $C_2 \setminus C_1$ appearing around $C_1$.

[3] A function is called *semi-algebraic* if its graph is semi-algebraic.

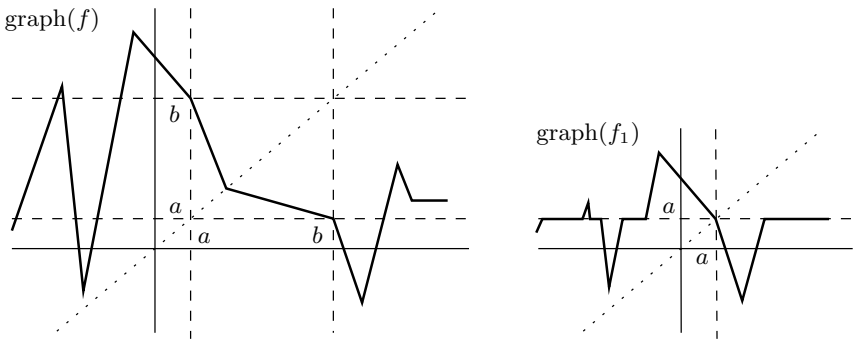



**Fig. 2.** Illustration of the construction of $f_1$ (right) from $f$ (left).

Sharkovskiĭ's theorem [1], one of the most fundamental result in dynamical system theory, tells us that for a continuous and terminating $f : \mathbb{R} \to \mathbb{R}$ only periods $1, 2, 4, \ldots, 2^d$ can appear for some integer value $d$. The previous lemma has the following corollary which strengthens the result of Sharkovskiĭ's for terminating functions.

**Corollary 2.** *If $f : \mathbb{R} \to \mathbb{R}$ is continuous and terminating, then $f$ can only have periodic points with periods 1 and 2.* □

Further on, in the proof of Theorem 3, for functions $f$ for which $Per(f)$ is $\mathbb{R}$, we only have to verify whether $\mathbb{R}$ is $C_1 \cup C_2$. For other $f$ we have to do further tests. Hereto, we now describe the construction of a continuous function $\tilde{f}$ from a given continuous function $f$.

Let $C$ denote the set $Per(f)$, which, by Lemma 2 and the assumption that $Per(f)$ is not $\mathbb{R}$, we can take to be $[a, b]$, $[a, +\infty)$ or $(-\infty, b]$.

First, we collapse $C$ to $\{a\}$ if $C$ is bounded or of the form $[a, +\infty)$; and to $\{b\}$ if $C$ is of the form $(-\infty, b]$. Let us first consider the case $C = [a, b]$. Let $f_{\in C} = \{x \in \mathbb{R} \mid f(x) \in C\}$, $f_{<C} = \{x \in \mathbb{R} \mid f(x) < a\}$, and $f_{>C} = \{x \in \mathbb{R} \mid f(x) > b\}$.

We define the continuous function $f_1$ on $\mathbb{R}$ as $f_1(x) :=$

$$\begin{cases} f(x) & \text{if } x \in f_{<C} \text{ and } x < a, \\ f(x) - (b - a) & \text{if } x \in f_{>C} \text{ and } x < a, \\ f(x + (b - a)) & \text{if } x \in f_{<C} \text{ and } x > a, \\ f(x + (b - a)) - (b - a) & \text{if } x \in f_{>C} \text{ and } x > a, \\ a & \text{if } x \in f_{\in C}. \end{cases}$$

This construction is illustrated in Fig. 2.

Let us next consider the case $C = [a, +\infty)$. Here, the function $f_1$ on $\mathbb{R}$ is defined as

$$\begin{cases} f(x) & \text{if } x \in f_{<C} \text{ and } x < a, \\ a & \text{if } x \in f_{\in C} \text{ and } x < a \text{ or if } x \geq a. \end{cases}$$

In the case $C = (-\infty, b]$, $f_1$ is defined similarly to the previous case. Finally, we define

$$\tilde{f}(x) := f_1(x + \alpha) - \alpha,$$

where $\alpha$ is $a$ or $b$, depending on the case.

The following lemma is readily verified.

**Lemma 5.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a function. We have $f^k(\mathbb{R}) = Per(f)$ if and only if $\tilde{f}^k(\mathbb{R}) = \{0\}$.* □

As mentioned in the previous section, in the area of dynamical systems, a function $\tilde{f}$ is called *nilpotent* if $\tilde{f}^k(\mathbb{R}) = \{0\}$ for some integer $k$. The following lemma's show that this is a decidable property in our setting. For continuous piecewise affine functions this result was already stated (without proof) [4]. So, we extend this result to continuous semi-algebraic functions and furthermore show that the decision procedure is expressible in FO.

**Lemma 6.** *There is an* FO *sentence that expresses whether a continuous semi-algebraic function $f : \mathbb{R} \to \mathbb{R}$ is nilpotent.*

Proof (sketch). We now describe the algorithm NILPOTENT(input $f$) to decide nilpotency of continuous semi-algebraic functions $f : \mathbb{R} \to \mathbb{R}$.

---

**Algorithm** NILPOTENT(input $f$):
*Step 1.* Compute the set $\{x \in \mathbb{R} \mid f^2(x) = x\}$. If this set differs from $\{0\}$, then answer *no*, else continue with Step 2.

*Step 2.* Compute the set $B = \{r \mid \gamma_{BB}(r)\}$, where $\gamma_{BB}(r)$ is the formula that defines positive real numbers $r$ that satisfy one of the following three conditions:

1. $\lim_{x\to-\infty} f(x)$ and $\lim_{x\to+\infty} f(x)$ are constants and $f((-\infty, r]) \subset (-r, +r)$ and $f([r, +\infty)) \subset (-r, +r)$;
2. $\lim_{x\to-\infty} f(x) = +\infty$ and $\lim_{x\to+\infty} f(x)$ is a constant and $f([r, +\infty)) \subset (-r, +r)$;
3. $\lim_{x\to-\infty} f(x)$ is a constant and $\lim_{x\to+\infty} f(x) = -\infty$ and $f((-\infty, r]) \subset (-r, +r)$;

If $B$ is empty, answer *no*, else compute the infimum $r_0$ of $B$ and continue with Step 3.

*Step 3.* Let $g$ be the function defined as $g(x) := f(x)$ if $-r_0 < x < r_0$ and $g(x) := f(-r_0)$ if $x \leq -r_0$ and $g(x) := f(r_0)$ if $x \geq r_0$. Take $r_1$ larger than $r_0$ and $\max\{|g(x)| \mid x \in \mathbb{R}\}$.

If for $g$ there exists a positive real number $\varepsilon$ such that

1. $g$ is constant 0 on $(-\varepsilon, +\varepsilon)$, or
2. $g$ is constant 0 on $(-\varepsilon, 0)$ and has a right tangent with strictly negative slope in 0, or
3. $g$ is constant 0 on $(0, +\varepsilon)$ and has a left tangent with strictly negative slope in 0,

then continue with Step 4, else answer *no*.

*Step 4.* If for all $x > 0$, $g(x) < x$ and $g^2(x) < x$ and for every $x < 0$, $g(x) > x$ and $g^2(x) > x$ holds, then answer *yes*, else answer *no*.

---

We omit the proof of the correctness of the algorithm NILPOTENT. □

We are now ready for the proof of Theorem 3.

PROOF OF THEOREM 3 (sketch). We describe a decision procedure TERMINATE that on input a function $f : \mathbb{R} \to \mathbb{R}$, decides whether the transitive closure of graph($f$) terminates after a finite number of iterations.

---

**Algorithm** TERMINATE(input $f$):
*Step 1.* Compute the sets $C_1 = \{x \mid f(x) = x\}$ and $C_2 = \{x \mid f^2(x) = x\}$. If $C_2$ is a closed and connected part of $\mathbb{R}$ and if $C_1$ is a point with $C_2 \setminus C_1$ around it or if $C_2 \setminus C_1$ is empty, then continue with Step 2, else answer *no*.

*Step 2.* If $C_2$ is $\mathbb{R}$, answer *yes*, else compute the function $\tilde{f}$ (as described before Lemma 5) and apply the algorithm NILPOTENT in the proof of Lemma 6 to $\tilde{f}$ and return the answer.

---

The correctness of this procedure follows from Lemmas 4, 5 and 6. It should be clear that all ingredients can be expressed in FO. □

We use the function $f_1$, given on the left in Fig. 1 in the Introduction, and the function $f_2$, given on the right in Fig. 1, to illustrate the decision procedure TERMINATE(input $f$). For $f_1$, $C_1 \cup C_2$ is $\{0, 1\}$, and therefore $f_1$ doesn't survive Step 1 and TERMINATE(input $f_1$) immediately returns *no*. For $f_2$, $C_1 \cup C_2$ is $\{0\}$, and therefore we have $\tilde{f}_2 = f_2$. Next, the algorithm NILPOTENT is called with input $f_2$. For $f_2$, the set $B$, computed in Step 2 of the algorithm NILPOTENT, is non-empty and $r_0$ is 1. So, the function $g$ in Step 3 will be $f_2$ again and $r_1$ is 1. Since $g$ is identical zero around the origin, finally the test in Step 4 decides. Here, we have that for $x > 0$, $g(x) < x$ and also $g^2(x) < x$ since $x - \frac{1}{4} < x$ and $x - \frac{1}{2} < x$. For $x < 0$, we have that both $g(x)$ and $g^2(x)$ are identical zero and thus the test succeeds also here. The output of NILPOTENT on input $f_2$ and therefore also the output of TERMINATE on input $f_2$ is *yes*.

For a continuous and terminating function, the periods that can appear are 1 and 2 (see Lemma 3). In dynamical systems theory, finding an upper bound on the length of the run-ups in terms of some characteristics of the function, is considered to be, even for piecewise affine functions, a difficult problem [18,20]. Take, for instance, the terminating continuous piecewise affine function that is constant towards $-\infty$ and $+\infty$ and that has turning points $(0, \frac{1}{3})$, $(\frac{1}{3}, \frac{2}{3} - \varepsilon)$, $(\frac{4}{9}, \frac{4}{9})$, $(\frac{5}{9}, \frac{5}{9})$, $(\frac{2}{3}, \frac{1}{3})$, and $(1, \frac{2}{3})$, with $\varepsilon > 0$ small. Here, it seems extremely difficult to find an upper bound on the length of the run-ups in terms of the number of line segments or of their endpoints. The best we can say in this context, is that from Theorem 3, it follows that also the maximal run-up can be computed.

**Corollary 3.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous, terminating semi-algebraic function. The maximal run-up of a point in $\mathbb{R}$ with respect to $f$ is computable.* □

We end this section with a remark concerning termination of continuous functions that are defined on a connected part $I$ of $\mathbb{R}$. Let $f : I \to I$ be such a function. We define the function $\bar{f}$ to be the continuous extension of $f$ to $\mathbb{R}$ that is constant on $\mathbb{R} \setminus I$. It is readily verified that the transitive closure of graph($f$) terminates if and only if $\bar{f}$ is terminating. We therefore have the following corollary of Theorem 3.

**Corollary 4.** *Let $I$ be a connected part of $\mathbb{R}$. There is an* FO *expressible decision procedure that decides whether the transitive closure of the graph of a continuous semi-algebraic function $f : I \to I$ terminates.* □

## 5 Logics with Transitive Closure Restricted to Function Graphs

In this section, we study fragments of FO+TC and FO+TCS where the transitive-closure operator is restricted to work only on the graphs of continuous semi-algebraic functions from $\mathbb{R}^k$ to $\mathbb{R}^k$. These languages bear some similarity with *deterministic* transitive-closure logics in finite model theory [8].

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are $k$-dimensional real vectors and if $\psi(\boldsymbol{x}, \boldsymbol{y})$ is an FO+TC-formula (respectively FO+TCS-formula), let $\gamma_\psi$ be the FO+TC-sentence (respectively FO+TCS-sentence) $\gamma_\psi^1 \wedge \gamma_\psi^2$, where $\gamma_\psi^1$ expresses that $\psi(\boldsymbol{x}, \boldsymbol{y})$ defines the graph of a function from $\mathbb{R}^k$ to $\mathbb{R}^k$ and where $\gamma_\psi^2$ expresses that $\psi(\boldsymbol{x}, \boldsymbol{y})$ defines a continuous function graph. We can express $\gamma_\psi^2$ using the classical $\varepsilon$-$\delta$ definition of continuity. Therefore, it should be clear that $\gamma_\psi^1$ and $\gamma_\psi^2$ can be expressed by formulas that make direct calls to $\psi(\boldsymbol{x}, \boldsymbol{y})$. Thus, the following property is readily verified.

**Proposition 1.** *Let $\psi(\boldsymbol{x}, \boldsymbol{y})$ be an* FO+TC*-formula (respectively an* FO+TCS*-formula). The evaluation of $\psi(\boldsymbol{x}, \boldsymbol{y})$ on an input database $A$ terminates if and only if the evaluation of $\gamma_\psi$ on $A$ terminates.* □

**Definition 6.** We define FO+cTC (respectively FO+cTCS) to be the fragment of FO+TC (respectively FO+TCS) in which only TC-expressions of the form $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y}) \wedge \gamma_\psi](\boldsymbol{s}, \boldsymbol{t})$ (respectively $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x}, \boldsymbol{y}) \wedge \gamma_\psi \mid \sigma](\boldsymbol{s}, \boldsymbol{t})$) are allowed to occur. □

We again use superscript numbers to denote restrictions on the arities of the relations of which transitive closure can be taken.

### 5.1 Deciding Termination of the Evaluation of FO+cTC$^2$ Queries

Since, when $\psi(x, y)$ is $y = x + 1$, $\gamma_\psi$ is *true*, from the proof of Theorem 2 the following negative result follows.

**Corollary 5.** *It is undecidable whether a given formula in* FO+cTCS$^2$ *terminates on a given input database.* □

We remark that for this undecidability it is not needed that the transitive closure of continuous functions on an *unbounded* domain is allowed ($f(x) = x+1$ in the proof of Theorem 2). Even when, for example, only functions from $[0,1]$ to $[0,1]$ are allowed, we have undecidability. We can see this by modifying the proof of Theorem 2 as follows. For any polynomial $p(x_1, \ldots, x_{13})$, let $\sigma_p$ be the FO-expressible stop-condition:

$$(\exists x_1)\cdots(\exists x_{13})(\bigwedge_{i=1}^{13}((\exists y_i)(x_i y_i = 1 \wedge X(1,y_i)) \vee x_i = 0 \vee x_i = 1) \wedge p(x_1,\ldots,x_{13}) = 0).$$

Since, in consecutive iterations, the function $\bar{f}$, for $f : [0,1] \to [0,1]$, with $f(x) = \frac{x}{x+1}$, maps 1 to $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$, it is then easy to see that $p(x_1, \ldots, x_{13})$ having an integer solution is equivalent to

$$[\mathrm{TC}_{x;y}\,\mathrm{graph}(\bar{f}) \mid \sigma_p](s,t)$$

terminating (remark again that the $\gamma_{\mathrm{graph}(\bar{f})}$ is *true*).

The main result of this section is the following.

**Theorem 4.** *It is decidable whether a given formula in* FO+cTC$^2$ *terminates on a given input database. Moreover, this decision procedure is expressible in* FO+cTC$^2$.

Proof (sketch). Given a formula $\varphi$ in FO+cTC$^2$ and an input database $A$, we can decide whether the evaluation of $\varphi$ on $A$ terminates by first evaluating the deepest FO-formulas on which a TC-operator works on $A$ and then using Theorem 3 to decide whether the computation of transitive closure halts on this set. If it does not terminate, we answer *no*, else we compute the result and continue recursively to less deep occurrences of TC-operators in $\varphi$. We continue this until the complete formula $\varphi$ is processed. Only if we reach the end and all intermediate termination tests returned *yes*, we output *yes*.

The expressibility of the decision procedure in FO can also be proven by induction on the structure of the formula. □

### 5.2 A Guarded Fragment of FO+cTC$^2$

The fact that termination of FO+cTC$^2$-formulas is expressible in FO+cTC$^2$, allows us to define a *guarded* fragment, FO+cTC$^2_{\mathcal{G}}$, of this language. Indeed, if $\psi$ is a formula in FO+cTC$^2$ of the form $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\,\psi(\boldsymbol{x},\boldsymbol{y})](\boldsymbol{s},\boldsymbol{t})$, let $\tau_\psi$ be the FO+cTC$^2$-sentence that expresses that this TC-expression terminates (obviously, $\tau_\psi$ also depends on the input database). We can now define the guarded fragment of FO+cTC$^2$, in which every TC-expression is accompanied by a *termination guard*.

**Definition 7.** We define FO+cTC$^2_{\mathcal{G}}$ to be the fragment of FO+cTC$^2$ in which only TC-expressions of the form $[\mathrm{TC}_{\boldsymbol{x};\boldsymbol{y}}\, \psi(\boldsymbol{x},\boldsymbol{y}) \wedge \tau_\psi](\boldsymbol{s},\boldsymbol{t})$ are allowed. □

The following property follows from the above remarks.

**Proposition 2.** *In the language* FO+cTC$^2_{\mathcal{G}}$*, every query terminates on all possible input databases. Furthermore, all terminating queries of* FO+cTC$^2$ *are expressible in* FO+cTC$^2_{\mathcal{G}}$*.* □

### 5.3 Expressiveness Results

Even the least expressive of the transitive-closure logics is more expressive than first-order logic.

**Theorem 5.** *The language* FO+cTC$^2_{\mathcal{G}}$ *is more expressive than* FO *on finite constraint databases.*

Proof (sketch). Consider the following query $Q_{\mathrm{int}}$ on 1-dimensional databases $S$: "Is $S$ a singleton that contains a natural number?". The query $Q_{\mathrm{int}}$ is not expressible in FO (if it would be expressible, then also the predicate $\mathrm{nat}(x)$, expressing that $x$ is a natural number, would be in FO). The query $Q_{\mathrm{int}}$ is expressible in FO+cTC$^2_{\mathcal{G}}$ by the sentence that says that $S$ is a singleton that contains 0, 1 or an element $r > 1$ such that $(\exists s)(\exists t)([\mathrm{TC}_{x;y}\, \psi(x,y) \wedge \gamma_\psi \wedge \tau_{\psi(x,y)\wedge\gamma_\psi}](s,t) \wedge s = 1 \wedge t = \frac{1}{r})$, where $\psi(x,y)$ is the formula $(\exists r)(S(r) \wedge \varphi(r,x,y))$. Here, $\varphi(r,x,y)$ defines the graph of the continuous piecewise affine function that maps $x$ to

$$y = \begin{cases} 0 & \text{if } x \leq \frac{1}{r}, \\ x - \frac{1}{r} & \text{if } \frac{1}{r} < x < 1, \\ 1 - \frac{1}{r} & \text{if } x \geq 1. \end{cases}$$

Remark that $\gamma_\psi$ is always *true*. The sentence $\tau_{\psi(x,y)\wedge\gamma_\psi}$ is *true* when the database is a singleton containing a number larger than one. The function given by $\varphi(r,x,y)$ is illustrated on the right in Fig. 1 for $r = 4$. The evaluation of this transitive closure is guaranteed to end after at most $\lceil r \rceil$ iterations and this sentence indeed expresses $Q_{\mathrm{int}}$ since $(1, \frac{1}{r})$ belongs to the result of the transitive closure if and only if $r > 1$ is a natural number. □

## 6 Concluding Remarks

We conclude with a number of remarks. One of our initial motivations to look at termination of query evaluation in transitive closure logics was to study the expressive power of FO+TC compared to that of FO+TCS. As mentioned in the Introduction, the latter language is computationally complete on linear constraint databases. It is not clear whether FO+TC is also complete. In general, we have no way to separate these languages. But if we restrict ourselves to their fragments FO+cTC$^2$ and FO+cTCS$^2$, the fact that for the former termination is decidable, whereas it is not for the latter, might give the impression that at least

these fragments can be separated. But this is not the case, since equivalence of formulas in these languages is undecidable. In fact, the expressions used in the proof of Theorem 2, are expressible in FO+TC (they do not even use an input database).

A last remark concerns the validity of the results in Section 4 for more general settings. Lemmas 1–5 are also valid for arbitrary real closed fields $R$. One could ask whether the same is true for Lemma 6. However, the proof of the correctness of the FO-sentence which decides global convergence in Step 4 [3], relies on the Bolzano-Weierstrass theorem, which is known not to be valid for arbitrary real closed fields [5]. Furthermore, we can even prove that no FO-sentence exists that decides termination of semi-algebraic functions $f : R \to R$ for arbitrary real closed fields $R$.

**Acknowledgments.** We thank the reviewers for a number of suggestions to improve the presentation of our results.

## References

1. Ll. Alsedà, J. Llibre, and M. Misiurewicz. *Combinatorial Dynamics and Entropy in Dimension One*, volume 5 of *Advances Series in Nonlinear Dynamics*. World Scientific, 1993.
2. M. Benedikt, M. Grohe, L. Libkin, and L. Segoufin. Reachability and connectivity queries in constraint databases. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'00)*, pages 104–115. ACM, 2000.
3. V.D. Blondel, O. Bournez, P. Koiran, C.H. Papadimitriou, and J.N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theoretical Computer Science*, 255(1-2):687–696, 2001.
4. V.D. Blondel, O. Bournez, P. Koiran, and J.N. Tsitsiklis. The stability of saturated linear dynamical systems is undecidable. *Journal of Computer and System Sciences*, 62(3):442–462, 2001.
5. J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*, volume 36 of *Ergebenisse der Mathematik und ihrer Grenzgebiete. Folge 3*. Springer-Verlag, 1998.
6. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer-Verlag, 1975.
7. M. Davis, Y. Matijasevič, and J. Robinson. Hilbert's Tenth Problem. Diophantine equations: positive aspects of a negative solution. In *Mathematical Developments Arising from Hilbert Problems*, volume 28, pages 323–378. American Mathematical Society, 1976.
8. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.
9. F. Geerts. Linear approximation of semi-algebraic spatial databases using transitive closure logic, in arbitrary dimension. In G. Ghelli and G. Grahne, editors, *Proceedings of the 8th International Workshop on Database Programming Languages (DBPL'01)*, volume 2397 of *Lecture Notes in Computer Science*, pages 182–197. Springer-Verlag, 2002.

10. F. Geerts and B. Kuijpers. Linear approximation of planar spatial databases using transitive-closure logic. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'00)*, pages 126–135. ACM, 2000.
11. S. Grumbach and G. Kuper. Tractable recursion over geometric data. In G. Smolka, editor, *Proceedings of Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 450–462. Springer-Verlag, 1997.
12. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Science*, 51(1):26–52, 1995. A preliminary report appeared in the *Proceedings 9th ACM Symposium on Principles of Database Systems (PODS'90)*.
13. P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132:113–128, 1994.
14. S. Kreutzer. Fixed-point query languages for linear constraint databases. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'00)*, pages 116–125. ACM, 2000.
15. S. Kreutzer. Operational semantics for fixed-point logics on constraint databases. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'01)*, volume 2250 of *Lecture Notes in Computer Science*, pages 470–484. Springer-Verlag, 2001.
16. S. Kreutzer. Query languages for constraint databases: First-order logic, fixed-points, and convex hulls. In J. Van den Bussche and V. Vianu, editors, *Proceedings of 8th International Conference on Database Theory (ICDT'01)*, volume 1973 of *Lecture Notes in Computer Science*, pages 248–262. Springer-Verlag, 2001.
17. G.M. Kuper, J. Paredaens, and L. Libkin. *Constraint databases*. Springer-Verlag, 1999.
18. J. Llibre and C. Preston. Personal communication. 2002.
19. Y. Matiyasevich. *Hilbert's Tenth Problem*. The MIT Press, 1993.
20. C. Preston. *Iterates of Maps on an Interval*, volume 999 of *Lecture Notes in Mathematics*. Springer-Verlag, 1983.
21. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1951.

# Data Exchange: Semantics and Query Answering

Ronald Fagin[1], Phokion G. Kolaitis[2⋆], Renée J. Miller[3∗], and Lucian Popa[1]

[1] IBM Almaden Research Center
[2] UC Santa Cruz
[3] University of Toronto

**Abstract.** Data exchange is the problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible. In this paper, we address foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange. These issues arise because, given a source instance, there may be many target instances that satisfy the constraints of the data exchange problem. We give an algebraic specification that selects, among all solutions to the data exchange problem, a special class of solutions that we call *universal*. A universal solution has no more and no less data than required for data exchange and it represents the entire space of possible solutions. We then identify fairly general, and practical, conditions that guarantee the existence of a universal solution and yield algorithms to compute a canonical universal solution efficiently. We adopt the notion of "certain answers" in indefinite databases for the semantics for query answering in data exchange. We investigate the computational complexity of computing the certain answers in this context and also study the problem of computing the certain answers of target queries by simply evaluating them on a canonical universal solution.

## 1 Introduction

In *data exchange*, data structured under one schema (which we call a *source schema*) must be restructured and translated into an instance of a different schema (a *target schema*). Data exchange is used in many tasks that require data to be transferred between existing, independently created applications. The first systems supporting the restructuring and translation of data were built several decades ago. An early such system was EXPRESS [21], which performed data exchange between hierarchical schemas. The need for systems supporting data exchange has persisted over the years. Recently this need has become more pronounced, as the terrain for data exchange has expanded with the proliferation of web data that are stored in different formats, such as traditional relational database schemas, semi-structured schemas (for example, DTDs or XML schemas), and various scientific formats. In this paper, we address several foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange.

⋆ Research carried out while these authors were visiting scientists at the IBM Almaden Research Center. Kolaitis was also partially supported by NSF Grant IIS-9907419. Miller was also partially supported by a research grant from NSERC.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 207–224, 2003.

**The data exchange problem.** In a data exchange setting, we have a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, where we assume that $\mathbf{S}$ and $\mathbf{T}$ are disjoint. Since $\mathbf{T}$ can be an independently created schema, it may have its own constraints that are given as a set $\Sigma_t$ of sentences in some logical formalism over $\mathbf{T}$. In addition, we must have a way of modeling the relationship between the source and target schemas. This essential element of data exchange is captured by *source-to-target dependencies* that specify how and what source data should appear in the target. These dependencies are assertions between a source query and a target query. Formally, we have a set $\Sigma_{st}$ of *source-to-target dependencies* of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula in some logical formalism over $\mathbf{S}$ and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula in some (perhaps different) logical formalism over $\mathbf{T}$.

Consider a data exchange setting determined by $\mathbf{S}$, $\mathbf{T}$, $\Sigma_{st}$, and $\Sigma_t$ as above. This setting gives rise to the following *data exchange problem:* given an instance $I$ over the source schema $\mathbf{S}$, materialize an instance $J$ over the target schema $\mathbf{T}$ such that the target dependencies $\Sigma_t$ are satisfied by $J$, and the source-to-target dependencies $\Sigma_{st}$ are satisfied by $I$ and $J$ together. The first crucial observation is that there may be many solutions (or none) for a given instance of the data exchange problem. Hence, several conceptual and technical questions arise concerning the semantics of data exchange. First, when does a solution exist? If many solutions exist, which solution should we materialize and what properties should it have, so that it reflects the source data as accurately as possible? Finally, can such a "good" solution be efficiently computed?

We consider the semantics of the data exchange problem to be one of the two main issues in data exchange. We believe that the other main issue is query answering. Specifically, suppose that $q$ is a query over the target schema $\mathbf{T}$ and $I$ is an instance over the source schema $\mathbf{S}$. What does answering $q$ with respect to $I$ mean? Clearly, there is an ambiguity arising from the fact that, as mentioned earlier, there may be many solutions $J$ for $I$ and, as a result, different such solutions $J$ may produce different answers $q(J)$. This conceptual difficulty was first encountered in the context of *incomplete* or *indefinite* databases (see, for instance, [23]), where one has to find the "right" answers to a query posed against a set of "possible" databases. There is general agreement that in the context of incomplete databases, the "right" answers are the *certain* answers, that is, the answers that occur in the intersection of all $q(J)$'s, as $J$ varies over all "possible" databases. This notion makes good sense for data exchange as well, where the "possible" databases are the solutions $J$ for the instance $I$. We thus adopt the certain answers for the semantics of query answering in the data exchange setting and investigate the complexity of computing the certain answers in the data exchange setting. A related important question is whether the certain answers of a query can be computed by query evaluation on the "good" target instance that we chose to materialize.

**Data exchange vs. data integration.** Before describing our results on data exchange, we briefly compare and contrast data exchange with *data integration*. Following the terminology and notation in the recent overview [13], a *data integration system* is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the *global schema*, $\mathcal{S}$ is the *source schema*, and $\mathcal{M}$ is a set of *assertions* relating elements of the global schema with elements of the source schema. Both $\mathcal{G}$ and $\mathcal{S}$ are specified in suitable languages that may allow for the expression of various constraints. In this generality, a data exchange setting $\langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ can be thought of as a data integration system in which $\mathbf{S}$ is the source schema, $\mathbf{T}$ and $\Sigma_t$ form the global schema, and the source-to-target dependencies in $\Sigma_{st}$ are the assertions of the

data integration system. In practice, however, most data integration systems studied to date are either LAV (*local-as-view*) systems or GAV (*global-as-view*) systems [11,13, 14]. In a LAV system, each assertion in $\mathcal{M}$ relates one element of the source schema $\mathcal{S}$ to a query (a view) over the global schema $\mathcal{G}$; moreover, it is usually assumed that there are no target constraints ($\Sigma_t = \emptyset$). In a GAV system the reverse holds, that is, each assertion in $\mathcal{M}$ relates one element of the global schema $\mathcal{G}$ to a query (a view) over the source schema $\mathcal{S}$. Since the source-to-target dependencies $\Sigma_{st}$ relate a query over the source schema $\mathbf{S}$ to a query over the target schema $\mathbf{T}$, a data exchange setting is neither a LAV nor a GAV system. Instead, it can be thought of as a GLAV (*global-and-local-as-view*) system [10,13].

The above similarities notwithstanding, there are important differences between data exchange and data integration. As mentioned earlier, in data exchange scenarios, the target schema is often independently created and comes with its own constraints. In data integration, however, the global schema $\mathcal{G}$ is commonly assumed to be a reconciled, virtual view of a heterogeneous collection of sources and, as such, has no constraints. In fact, with the notable exception of [5], which studied the impact of key and foreign key constraints on query answering in a GAV system, most research on data integration has not taken target constraints into account. Still, a more significant difference is that in a data exchange setting we have to actually materialize a finite target instance that best reflects the given source instance. In data integration no such exchange of data is required. For query answering, both data exchange and data integration use the certain answers as the standard semantics of queries over the target (global) schema. In data integration, the source instances are used to compute the certain answers of queries over the global schema. In contrast, in a data exchange setting, it may not be feasible to couple applications together in a manner that data may be retrieved and shared on-demand at query time. This may occur, for instance, in peer-to-peer applications that must share data, yet maintain autonomy. Hence, queries over the target schema may have to be answered using the materialized target instance alone, without reference to the original source instance. This leads to the following problem in data exchange: under what conditions and for which queries can the certain answers be computed using just the materialized target instance?

**Motivation from Clio.** The results presented here were motivated by our experience with Clio, a prototype schema mapping and data exchange tool to whose development some of us have contributed [18,19]. In Clio, source-to-target dependencies are (semi)-automatically generated from a set of correspondences between the source schema and the target schema; these dependencies can then be used in a data integration system to compute the certain answers to target queries. Most of the applications we considered, however, were decoupled applications that would have had to be rewritten to operate cooperatively, as required in data integration. For this reason, early on in the development of Clio, we recognized the need to go farther and, given a source instance, generate a single "universal" target instance that was the result of the schema mapping. In designing the algorithms used in Clio for creating the target instance, we were guided mainly by our own intuition rather than by formal considerations. It should be noted that there is a long history of work on data translation that focuses on taking high-level, data-independent translation rules and generating efficient, executable translation programs [1,20,21]. Yet, we could not find a formal justification for the intuitive choices we made in creating

the target instance. In seeking to formalize this intuition and justify the choices made in Clio, we were led to explore foundational and algorithmic issues related to the semantics of data exchange and query answering in this setting. Clio supports schemas that are relational or semi-structured. However, challenging issues already arise in the relational case. For this reason, here we focus exclusively on data exchange between relational schemas; extending this work to other types of schemas is the subject of on-going investigation.

**Summary of Results.** In Section 2, we formally introduce the data exchange problem. We then give an algebraic specification that selects, among all possible solutions for a given source instance, a special class of solutions that we call *universal*. More precisely, a solution for an instance of the data exchange problem is universal if it has homomorphisms to all solutions for that instance. We show that a universal solution has "good" properties that justify its choice for the semantics of the data exchange problem. We note that Calì et al. [5] studied GAV systems with key and foreign key constraints at the target. By means of a logic program that simulates the foreign key constraints, they constructed a *canonical database*, which turns out to be a particular example of our notion of universal solution.

Given the declarative specification of universal solutions, we go on in Section 3 to identify fairly general, yet practical, sufficient conditions that guarantee the existence of a universal solution and yield algorithms to compute such a solution efficiently. We introduce the concept of a *weakly acyclic* set of target dependencies, which is broad enough to contain as special cases both sets of full tuple-generating dependencies (full tgds) [4] and acyclic sets of inclusion dependencies [7]. We then show that if $\Sigma_{st}$ is a set of tuple-generating dependencies (tgds) and $\Sigma_t$ is the union of a weakly acyclic set of tgds with a set of equality generating dependencies (egds), then, given a source instance of the data exchange problem, (1) a universal solution exists if and only if a solution exists, and (2) there is a polynomial-time algorithm that determines whether a solution exists and, if so, it produces a particular universal solution, which we call the *canonical* universal solution. These results make use of the classical *chase* procedure [4,15]. We note that, even though the chase has been widely used in reasoning about dependencies, we have not been able to find any explicit references to the fact that the chase can produce instances that have homomorphisms to all instances satisfying the dependencies under consideration.

After this, in Sections 4 and 5, we address algorithmic issues related to query answering in the data exchange setting. We study the computational complexity of computing the certain answers, and explore the boundary of what queries can and cannot be answered in a data exchange setting using the exchanged target instance alone. On the positive side, if $q$ is a union of conjunctive queries, then it is easy to show that the certain answers of $q$ can indeed be obtained by evaluating $q$ on an arbitrary universal solution. Moreover, universal solutions are the only solutions possessing this property; this can be seen as further justification for our choice to use universal solutions for data exchange. It also follows that, whenever a universal solution can be computed in polynomial time, the certain answers of unions of conjunctive queries can be computed in polynomial time (in particular, this is true when the dependencies in $\Sigma_{st}$ and $\Sigma_t$ satisfy the conditions identified in Section 3).

On the negative side, a dramatic change occurs when queries have inequalities. The hardness of this problem arises from the complexity of reasoning over uncertain databases, not from the data exchange *per se*. Indeed, Abiteboul and Duschka [2] showed that in a LAV data integration system and with conjunctive queries as views, computing the certain answers of conjunctive queries with inequalities is a coNP-complete problem. Since LAV is a special case of a data exchange setting in which the canonical universal solution can be computed in polynomial time, it follows that, unless $\mathrm{P} = \mathrm{NP}$, we cannot compute the certain answers of conjunctive queries with inequalities by evaluating them on the canonical universal solution (or on any other polynomial-time computable universal solution).

We take a closer look at conjunctive queries with inequalities by focusing on the number of inequalities. In [2], it was claimed that in the LAV setting and with conjunctive queries as views, computing the certain answers of conjunctive queries with a single inequality is a coNP-hard problem. The reduction given in that paper, however, is not correct; a different reduction in the unpublished full version [3] shows that computing the certain answers of conjunctive queries with six (or more) inequalities is a coNP-complete problem. We conjecture that the minimum number of inequalities that give rise to such coNP-hardness results is two. Towards this, we show that in the same LAV setting, computing the certain answers of *unions* of conjunctive queries with at most two inequalities per disjunct is coNP-complete. This result is tight, because we show that, even for the more general data exchange setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct (thus, the claim in [2] is false, unless $\mathrm{P} = \mathrm{NP}$). Moreover, the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed in time polynomial in the size of a given universal solution. We point out, however, that this computation cannot be carried out by simply evaluating such queries on the canonical universal solution. Thus, the question arises as to whether the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed by evaluating some other (perhaps more complex) first-order query on the canonical universal solution. Our final theorem provides a strong negative answer to this question. It shows that there is a simple conjunctive query $q$ with one inequality for which there is no first-order query $q^*$ such that the certain answers of $q$ can be computed by evaluating $q^*$ on the canonical universal solution. The proof of this theorem makes use of a novel combination of Ehrenfeucht-Fraïssé games and the chase.

The proofs of the results of this paper can be found in the full version [9].

## 2 The Data Exchange Problem

A *schema* is a finite collection $\mathbf{R} = \{R_1, \ldots, R_k\}$ of relation symbols. An *instance* $I$ *over the schema* $\mathbf{R}$ is a function that associates to each relation symbol $R_i$ a relation $I(R_i)$. set of relations $I(R_1), \ldots, I(R_k)$ interpreting the corresponding relation symbols in $\mathbf{S}$. In the sequel, we will on occasion abuse the notation and use $R_i$ to denote both the relation symbol and the relation that interprets it. Given a tuple $t$ occurring in a relation $R$, we denote by $R(t)$ the association between $t$ and $R$ and call it a *fact*. If $\mathbf{R}$ is a schema, then a *dependency over* $\mathbf{R}$ is a sentence in some logical formalism over $\mathbf{R}$.

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be two disjoint schemas. We refer to $\mathbf{S}$ as the *source* schema and to the $S_i$'s as the *source* relation symbols. We refer to $\mathbf{T}$ as the *target* schema and to the $T_j$'s as the *target* relation symbols. Similarly, instances over $\mathbf{S}$ will be called *source* instances, while instances over $\mathbf{T}$ will be called *target* instances. If $I$ is a source instance and $J$ is a target instance, then we write $\langle I, J\rangle$ for the instance $K$ over the schema $\mathbf{S} \cup \mathbf{T}$ such that $K(S_i) = I(S_i)$ and $K(T_j) = J(T_j)$, for $i \leq n$ and $j \leq m$.

A *source-to-target dependency* is a dependency of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, of some logical formalism over $\mathbf{S}$ and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, over some logical formalism over $\mathbf{T}$ (these two logical formalisms may be different). We use the notation $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$. A *target* dependency is a dependency over the target schema $\mathbf{T}$ (the formalism used to express a target dependency may be different from those used for the source-to-target dependencies). The source schema may also have dependencies that we assume are satisfied by every source instance. While the source dependencies may play an important role in deriving source-to-target dependencies [19], they do not play any direct role in data exchange, because we take the source instance to be given.

**Definition 1.** A *data exchange setting* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set $\Sigma_{st}$ of source-to-target dependencies, and a set $\Sigma_t$ of target dependencies. The *data exchange problem* associated with this setting is the following: given a finite source instance $I$, find a finite target instance $J$ such that $\langle I, J\rangle$ satisfies $\Sigma_{st}$ and $J$ satisfies $\Sigma_t$. Such a $J$ is called a *solution for* $I$ or, simply a *solution* if the source instance $I$ is understood from the context. The set of all solutions for $I$ is denoted by $\mathrm{Sol}(I)$.

For most practical purposes, and for most of the results of this paper, each source-to-target dependency in $\Sigma_{st}$ is a *tuple generating dependency (tgd)* [4] of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. Moreover, each target dependency in $\Sigma_t$ is either a *tuple-generating dependency (tgd)* (of the form shown below left) or an *equality-generating dependency (egd)* [4] (shown below right):

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})) \qquad \forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2))$$

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\mathbf{T}$, and $x_1$, $x_2$ are among the variables in $\mathbf{x}$. Note that data exchange settings with tgds as source-to-target dependencies include as special cases both LAV and GAV data integration systems in which the views are sound [13] and are defined by conjunctive queries. It is natural to take the target dependencies to be tgds and egds: these two classes together comprise the (embedded) implicational dependencies [8], which seem to include essentially all of the naturally-occurring constraints on relational databases. However, it is somewhat surprising that tgds, which were originally "designed" for other purposes (as constraints), turn out to be ideally suited for describing desired data transfer. For simplicity, in the rest of the paper we will drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, we will write down all existential quantifiers.

The next example shows that there may be more than one possible solution for a given data exchange problem. The natural question is then which solution to choose.

*Example 1.* Consider a data exchange problem in which the source schema has three relation symbols $P, Q, R$, each of them with attributes $A, B, C$, while the target schema has one relation symbol $T$ also with attributes $A, B, C$. We assume that $\Sigma_t = \emptyset$. The source-to-target dependencies and the source instance are:

$$\begin{array}{lll} \Sigma_{st}: & P(a,b,c) \rightarrow \exists Y \exists Z\ T(a,Y,Z) & I = \{\ \ P(a_0, b_0', c_0'), \\ & Q(a,b,c) \rightarrow \exists X \exists U\ T(X,b,U) & \qquad Q(a_0'', b_0, c_0''), \\ & R(a,b,c) \rightarrow \exists V \exists W\ T(V,W,c) & \qquad R(a_0''', b_0''', c_0)\ \} \end{array}$$

We observe first that the dependencies in $\Sigma_{st}$ do not completely specify the target instance. It should be noted that such incomplete specification arises naturally in many practical scenarios of data exchange (or data integration for that matter; see [11,13]). For our example, one possible solution is:

$$J = \{T(a_0, Y_0, Z_0), T(X_0, b_0, U_0), T(V_0, W_0, c_0)\},$$

where $X_0, Y_0, \ldots$ represent "unknown" values. We will call such values *labeled nulls* and we will introduce them formally in the next section. The second observation is that there may be more than one solution. For example, the following are solutions as well:

$$J_1 = \{T(a_0, b_0, c_0)\} \qquad J_2 = \{T(a_0, b_0, Z_1), T(V_1, W_1, c_0)\}$$

In the above, $Z_1, V_1$ and $W_1$ are labeled nulls. Note that $J_1$ does not use labeled nulls; instead, source values are used to witness the existentially quantified variables in the dependencies. Solution $J_1$ seems to be less general than $J$, since it "assumes" that all three tuples required by the dependencies are equal to the tuple $(a_0, b_0, c_0)$. This assumption, however, is not part of the specification. Similarly, solution $J_2$ has extra information that is not a consequence of the dependencies in $\Sigma_{st}$ for the given source data. We argue that neither $J_1$ nor $J_2$ should be used for data exchange. In contrast, $J$ is the "best" solution: it contains no more and no less than what the specification requires. We formalize this intuition next.

### 2.1 Universal Solutions

We next give an algebraic specification that selects, among all possible solutions, a special class of solutions that we call *universal*. As we will see, a universal solution has several "good" properties that justify its choice for the semantics of data exchange. Before presenting the key definition, we introduce some terminology and notation.

We denote by $\underline{\mathsf{Const}}$ the set of all values that occur in source instances and we also call them *constants*. In addition, we assume an infinite set $\underline{\mathsf{Var}}$ of values, which we call *labeled nulls*, such that $\underline{\mathsf{Var}} \cap \underline{\mathsf{Const}} = \emptyset$. We reserve the symbols $I, I', I_1, I_2, \ldots$ for instances over the source schema $I$ and with values in $\underline{\mathsf{Const}}$. We also reserve the symbols $J, J', J_1, J_2, \ldots$ for instances over the target schema $\mathbf{T}$ and with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

If $\mathbf{R} = \{R_1, \ldots, R_k)$ is a schema and $K$ is an instance over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$, then $\underline{\mathsf{Var}}(K)$ denotes the set of labelled nulls occurring in relations in $K$.

**Definition 2.** Let $K_1$ and $K_2$ be two instances over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

A *homomorphism* $h : K_1 \rightarrow K_2$ is a mapping from $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_1)$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\mathsf{Const}}$; (2) for every fact $R_i(t)$ of $K_1$, we have that $R_i(h(t))$ is a fact of $K_2$ (where, if $t = (a_1, \ldots, a_s)$, then $h(t) = (h(a_1), \ldots, h(a_s))$).

$K_1$ is *homomorphically equivalent* to $K_2$ if there is a homomorphism $h : K_1 \rightarrow K_2$ and a homomorphism $h' : K_2 \rightarrow K_1$.

**Definition 3 (Universal solution).** Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$. If $I$ is a source instance, then a *universal solution for* $I$ is a solution $J$ for $I$ such that for every solution $J'$ for $I$, there exists a homomorphism $h : J \rightarrow J'$.

*Example 2.* The instances $J_1$ and $J_2$ in Example 1 are not universal. In particular, there is no homomorphism from $J_1$ to $J$ and also there is no homomorphism from $J_2$ to $J$. This fact makes precise our earlier intuition that the instances $J_1$ and $J_2$ contain "extra" information. In contrast, there exist homomorphisms from $J$ to both $J_1$ and $J_2$. Actually, it can be easily shown that $J$ has homomorphisms to all other solutions. Thus, $J$ is universal.

From an algebraic standpoint, being a universal solution is a property akin to being an *initial structure* [17] for the set of all solutions (although an initial structure for a set $\mathcal{K}$ of structures is required to have *unique* homomorphisms to all other structures in $\mathcal{K}$). Initial structures are ubiquitous in several areas of computer science, including semantics of programming languages and term rewriting, and are known to have good properties (see [17]). The next result asserts that universal solutions have good properties as well.

**Proposition 1.** *Let* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ *be a data exchange setting.*

1. *If* $I$ *is a source instance and* $J$, $J'$ *are universal solutions for* $I$, *then* $J$ *and* $J'$ *are homomorphically equivalent.*
2. *Assume that* $\Sigma_{st}$ *is a set of tgds. Let* $I$, $I'$ *be two source instances,* $J$ *a universal solution for* $I$, *and* $J'$ *a universal solution for* $I'$. *Then* $\mathrm{Sol}(I) \subseteq \mathrm{Sol}(I')$ *if and only if there is a homomorphism* $h : J' \rightarrow J$. *Consequently,* $\mathrm{Sol}(I) = \mathrm{Sol}(I')$ *if and only if* $J$ *and* $J'$ *are homomorphically equivalent.*

The first part of Proposition 1 asserts that universal solutions are unique up to homomorphic equivalence. The second part implies that if $J$ is a universal solution for two source instances $I$ and $I'$, then $\mathrm{Sol}(I) = \mathrm{Sol}(I')$. Thus, in a certain sense, each universal solution precisely embodies the space of solutions.

## 3 Computing Universal Solutions

Checking the conditions in Definition 3 requires implicitly the ability to check the (infinite) space of all solutions. Thus, it is not clear, at first hand, to what extent the notion of universal solution is a computable one. This section addresses the question of how

to check the existence of a universal solution and how to compute one (if one exists). In particular, we show that the classical chase can be used for data exchange and that every finite chase, if it does not fail, constructs a universal solution. If the chase fails, then no solution exists. However, in general, for arbitrary dependencies, there may not exist a finite chase. Hence, in Section 3.2 we introduce the class of weakly acyclic sets of tgds, for which the chase is guaranteed to terminate in polynomial time. For such dependencies, we show that: (1) the existence of a universal solution can be checked in polynomial time, (2) a universal solution exists if and only if a solution exists, and (3) a universal solution (if solutions exist) can be produced in polynomial time.

### 3.1 Chase: Canonical Generation of Universal Solutions

Intuitively, we apply the following procedure to produce a universal solution: start with an instance $\langle I, \emptyset \rangle$ that consists of $I$, for the source schema, and of the empty instance, for the target schema; then chase $\langle I, \emptyset \rangle$ by applying the dependencies in $\Sigma_{st}$ and $\Sigma_t$ for as long as they are applicable[1]. This process may fail (as we shall see shortly, if an attempt to identify two constants is made) or it may never terminate. But if it does terminate and if it does not fail, then the resulting instance is guaranteed to satisfy the dependencies and, moreover, to be universal (Theorem 1).

We next define chase steps. Similar to homomorphisms between instances, a homomorphism from a conjunctive formula $\phi(\mathbf{x})$ to an instance $J$ is a mapping from the variables $\mathbf{x}$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(J)$ such that for every atom $R(x_1, \ldots, x_n)$ of $\phi$, the fact $R(h(x_1), \ldots, h(x_n))$ is in $J$. The chase that we use is a slight variation of the classical notion of chase with tgds and egds of [4] in the sense that we chase instances rather than symbolic tableaux. Consequently, the chase may fail.

**Definition 4 (Chase step).** Let $K$ be an instance.

**(tgd)** Let $d$ be a tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$. Let $h$ be a homomorphism from $\phi(\mathbf{x})$ to $K$ such that there is no extension of $h$ to a homomorphism $h'$ from $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to $K$. We say that $d$ *can be applied to* $K$ *with homomorphism* $h$.

Let $K'$ be the union of $K$ with the set of facts obtained by: (a) extending $h$ to $h'$ such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by (b) taking the image of the atoms of $\psi$ under $h'$. We say that *the result of applying* $d$ *to* $K$ *with* $h$ is $K'$, and write $K \xrightarrow{d,h} K'$.

**(egd)** Let $d$ be an egd $\phi(\mathbf{x}) \rightarrow (x_1 = x_2)$. Let $h$ be a homomorphism from $\phi(\mathbf{x})$ to $K$ such that $h(x_1) \neq h(x_2)$. We say that $d$ *can be applied to* $K$ *with homomorphism* $h$. We distinguish two cases.

- If both $h(x_1)$ and $h(x_2)$ are in $\underline{\mathsf{Const}}$ then we say that *the result of applying* $d$ *to* $K$ *with* $h$ is "failure", and write $K \xrightarrow{d,h} \perp$.
- Otherwise, let $K'$ be $K$ where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the labeled null is replaced everywhere by the constant; if both are labeled nulls, then one is replaced everywhere by the other. We say that *the result of applying* $d$ *to* $K$ *with* $h$ is $K'$, and write $K \xrightarrow{d,h} K'$.

[1] It is possible to apply first $\Sigma_{st}$ as long as applicable and then apply $\Sigma_t$ as long as applicable.

In the definition, $K \xrightarrow{d,h} K'$ (including the case where $K'$ is $\perp$) defines one single chase step. We next define chase sequences and finite chases.

**Definition 5 (Chase).** Let $\Sigma$ be a set of tgds and egds, and let $K$ be an instance.

- A *chase sequence of $K$ with $\Sigma$* is a sequence (finite or infinite) of chase steps $K_i \xrightarrow{d_i,h_i} K_{i+1}$, with $i = 0, 1, \ldots$, with $K = K_0$ and $d_i$ a dependency in $\Sigma$.
- A *finite chase of $K$ with $\Sigma$* is a finite chase sequence $K_i \xrightarrow{d,h} K_{i+1}, 0 \leq i < m$, with the requirement that either (a) $K_m = \perp$ or (b) there is no dependency $d_i$ of $\Sigma$ and there is no homomorphism $h_i$ such that $d_i$ can be applied to $K_m$ with $h_i$. We say that $K_m$ is the result of the finite chase. We refer to case (a) as the case of a *failing finite chase* and we refer to case (b) as the case of a *successful finite chase*.

In general, there may not exist a finite chase of an instance (cyclic sets of dependencies could cause infinite application of chase steps). Infinite chases can be defined as well, but for this paper we do not need to do so. Also, different chase sequences may yield different results. However, each result, if not $\perp$, satisfies $\Sigma$.

For data exchange, we note first that, due to the nature of our dependencies, any chase sequence that starts with $\langle I, \emptyset \rangle$ does not change or add tuples in $I$. Then, if a finite chase exists, its result $\langle I, J \rangle$ is such that $J$ is a solution. Furthermore, $J$ is universal, a fact that does not seem to have been explicitly noted in the literature on the chase. The next theorem states this, and also states that the chase can be used to check the existence of a solution.

**Theorem 1.** *Assume a data exchange setting where $\Sigma_{st}$ consists of tgds and $\Sigma_t$ consists of tgds and egds.*

1. *Let $\langle I, J \rangle$ be the result of some successful finite chase of $\langle I, \emptyset \rangle$ with $\Sigma_{st} \cup \Sigma_t$. Then $J$ is a universal solution.*
2. *If there exists some failing finite chase of $\langle I, \emptyset \rangle$ with $\Sigma_{st} \cup \Sigma_t$, then there is no solution.*

For case 1 of Theorem 1 we refer to such $J$ as *a canonical universal solution*. In further examples and proofs, when such $J$ is unique, we will also use the term *the canonical universal solution*. We note that a canonical universal solution is similar, in its construction, to the representative instance defined in the work on the universal relation (see [16]).

The following is an example of cyclic set of inclusion dependencies for which there is no finite chase; thus, we cannot produce a universal solution by the chase. Still, a finite solution does exist. This illustrates the need for introducing restrictions in the class of dependencies that are allowed in the target.

*Example 3.* Consider the data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ as follows. The source schema $\mathbf{S}$ has one relation $\texttt{DeptEmp}(\texttt{dpt\_id}, \texttt{mgr\_name}, \texttt{eid})$ listing departments with their managers and their employees. The target schema $\mathbf{T}$ has a relation $\texttt{Dept}(\texttt{dpt\_id}, \texttt{mgr\_id}, \texttt{mgr\_name})$ for departments and their managers, and a separate relation for employees $\texttt{Emp}(\texttt{eid}, \texttt{dpt\_id})$. The source-to-target and target dependencies are:

$$\Sigma_{st} = \{\, \texttt{DeptEmp}(d, n, e) \rightarrow \exists M.\texttt{Dept}(d, M, n) \wedge \texttt{Emp}(e, d) \,\}$$
$$\Sigma_t = \{\, \texttt{Dept}(d, m, n) \rightarrow \exists D.\texttt{Emp}(m, D), \quad \texttt{Emp}(e, d) \rightarrow \exists M \exists N.\texttt{Dept}(d, M, N) \,\}$$

Assume now that the source instance $I$ has one tuple in DeptEmp, for department *CS* with manager *Mary* and employee *E003*. Chasing $\langle I, \emptyset \rangle$ with $\Sigma_{st}$ yields the target instance:

$$J_1 = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS)\}$$

where $M$ is a labeled null that instantiates the existentially quantified variable of the tgd, and encodes the unknown manager id of *Mary*. However, $J_1$ does not satisfy $\Sigma_t$; therefore, the chase does not stop at $J_1$. The first tgd in $\Sigma_t$ requires $M$ to appear in Emp as an employee id. Thus, the chase will add $\texttt{Emp}(M, D)$ where $D$ is a labeled null representing the unknown department in which Mary is employeed. Then the second tgd becomes applicable, and so on. It is easy to see that there is no finite chase. Satisfying all the dependencies would require building an infinite instance:

$$J = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS), \texttt{Emp}(M, D), \texttt{Dept}(D, M', N'), \dots\}$$

On the other hand, finite solutions exist. Two such examples are:

$$J' = \{\texttt{Dept}(CS, E003, Mary), \texttt{Emp}(E003, CS)\}$$
$$J'' = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS), \texttt{Emp}(M, CS)\}$$

However, neither $J'$ nor $J''$ are universal: there is no homomorphism from $J'$ to $J''$ and there is no homomorphism from $J''$ to $J'$. We argue that neither should be used for data exchange. In particular, $J'$ makes the assumption that the manager id of *Mary* is equal to *E003*, while $J''$ makes the assumption that the department in which *Mary* is employed is the same as the department (*CS*) that *Mary* manages. Neither assumption is a consequence of the given dependencies and source instance. It can be shown that no *finite* universal solution exists for this example.

We next consider sets of dependencies for which every chase sequence is guaranteed to reach its end after at most polynomially many steps (in the size of the input instance). For such sets of dependencies it follows that checking the existence of a solution, as well as generating a universal solution, can be carried out in polynomial time.

### 3.2 Polynomial-Length Chase

We first discuss sets of *full tgds* (tgds with no existentially quantified variables). It has been proven in [4] that every chase sequence with a set $\Sigma$ of full tgds has at most finite length. Moreover every chase has the same result. It is simple to show that the length of the chase is bounded by a polynomial in the size of the input instance (the dependencies and the schema are fixed). Also, any set of egds can be added to $\Sigma$ without affecting the uniqueness of the result or the polynomial bound.

Although full tgds enjoy nice properties, they are not very useful in practice. Most dependencies occurring in real schemas are non-full, for example, foreign key constraints or, more generally, inclusion dependencies [6]. It is well known that chasing with inclusion dependencies may not terminate in general. *Acyclic sets of inclusion dependencies* [7] are a special case for which every chase sequence has a length that is polynomial in the size of the input instance. Such dependencies can be described by defining a directed graph in which the nodes are the relation symbols, and such that

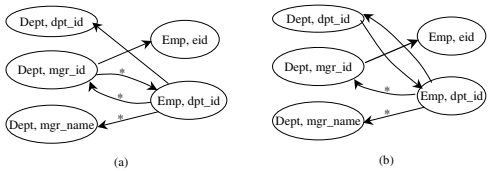



**Fig. 1.** Dependency graphs for: (a) a set of tgds that is not weakly acyclic, (b) a weakly acyclic set of tgds.

there exists an edge from $R$ to $S$ whenever there is an inclusion dependency from $R$ to $S$. A set of inclusion dependencies is acyclic if there is no cycle in this graph. We define next a *weakly acyclic sets of tgds*, a notion that strictly includes both sets of full tgds and acyclic sets of inclusion dependencies. This notion is inspired by the definition of weakly recursive ILOG [12], even though the latter is not directly related to dependencies. Informally, a set of tgds is weakly acyclic if it does not allow for cascading of labeled null creation during the chase.

**Definition 6 (Weakly acyclic set of tgds).** Let $\Sigma$ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair $(R, A)$ with $R$ a relation symbol of the schema and $A$ an attribute of $R$; call such pair $(R, A)$ a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in $\Sigma$ and for every $x$ in $\mathbf{x}$ that **occurs** in $\psi$:

- For every occurence of $x$ in $\phi$ in position $(R, A_i)$:
  (a) for every occurence of $x$ in $\psi$ in position $(S, B_j)$, add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).
  (b) in addition, for every existentially quantified variable $y$ and for every occurrence of $y$ in $\psi$ in position $(T, C_k)$, add a *special edge* $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$ (if it does not already exist).

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then $\Sigma$ is *weakly acyclic* if the dependency graph has no cycle going through a special edge. ∎

Intuitively, part (a) keeps track of the fact that a value may propagate from position $(R, A_i)$ to position $(S, B_j)$ during the chase. Part (b), moreover, keeps track of the fact that propagation of a value into $(S, B_j)$ also creates a labeled null in any position that has an existentially quantified variable. If a cycle goes through a special edge, then a labeled null appearing in a certain position during the chase may determine the creation of another labeled null, in the same position, at a later chase step. This process may thus continue forever. Note that the definition allows for cycles as long as they do not include special edges. In particular, a set of full tgds is a special case of a weakly acyclic set of tgds (there are no existentially quantified variables, and hence no special edges).

*Example 4.* Recall Example 3. The dependency graph of $\Sigma_t$ is shown in Figure 1(a). The graph contains a cycle with two special edges. Hence $\Sigma_t$ is not weakly acyclic and therefore a finite chase may not exist (as seen in Example 3). On the other hand, let us assume that we know that each manager of a department is employed by the *same* department. Then, we replace the set $\Sigma_t$ by the set $\Sigma'_t$, where

$$\Sigma_t' = \{ \texttt{Dept}(d, m, n) \rightarrow \texttt{Emp}(m, d),\ \texttt{Emp}(e, d) \rightarrow \exists M \exists N.\texttt{Dept}(d, M, N) \}$$

The dependency graph of $\Sigma_t'$, shown in Figure 1(b), has no cycles going through a special edge. Thus, $\Sigma_t'$ is weakly acyclic. As Theorem 2 will show, it is guaranteed that every chase sequence is finite. For Example 3, one can see that the chase of $J_1$ with $\Sigma_t'$ stops with result $J''$. Thus $J''$ is universal. Note that for $J''$ to be universal it was essential that we explicitly encoded in the dependencies the fact that managers are employeed by the department they manage. Finally, we remark that $\Sigma_t'$ is an example of a set of inclusion dependencies that, although weakly acyclic, is cyclic according to the definition of [7].

We now state the main results regarding weakly acyclic sets of tgds.

**Theorem 2.** *Let $\Sigma$ be the union of a weakly acyclic set of tgds with a set of egds, and let $K$ be an instance. Then there exists a polynomial in the size of $K$ that bounds the length of every chase sequence of $K$ with $\Sigma$.*

**Corollary 1.** *Assume a data exchange setting where $\Sigma_{st}$ is a set of tgds, and $\Sigma_t$ is the union of a weakly acyclic set of tgds with a set of egds. The existence of a solution can be checked in polynomial time. If a solution exists, then a universal solution can be produced in polynomial time.*

## 4 Query Answering

As stated earlier, we adopt the notion of certain answers for the semantics of query answering. We first give the formal definition of this notion and then address the problem of whether and to what extent the certain answers of a query over the target schema can be computed by evaluating some query (same or different) on a universal solution.

**Definition 7.** Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

- Let $q$ be a $k$-ary query over the target schema $\mathbf{T}$ and $I$ a source instance. The *certain answers of $q$ with respect to $I$*, denoted by *certain*$(q, I)$, is the set of all $k$-tuples $t$ of constants from $I$ such that for every solution $J$ of this instance of the data exchange problem, we have that $t \in q(J)$.
- Let $q$ be a Boolean query over the target schema $\mathbf{T}$ and $I$ a source instance. We write *certain*$(q, I) = \underline{\mathsf{true}}$ to denote that for every solution $J$ of this instance of the data exchange problem, we have that $q(J) = \underline{\mathsf{true}}$. We also write *certain*$(q, I) = \underline{\mathsf{false}}$ to denote that there is a solution $J$ such that $q(J) = \underline{\mathsf{false}}$.

On the face of it, the definition of certain answers entails a computation over the entire set of solutions of a given instance of the data exchange problem. Since this set may very well be infinite, it is desirable to identify situations in which the certain answers of a query $q$ can be computed by evaluating $q$ on a particular fixed solution and then keeping only the tuples that consist entirely of constants. More formally, if $q$ is a $k$-ary query and $J$ is a target instance, then $q(J)_{\downarrow}$ is the set of all $k$-tuples $t$ of constants such that $t \in q(J)$. We extend the notation to Boolean queries by agreeing that if $q$ is a Boolean query, then $q(J)_{\downarrow} = q(J)$ (= $\underline{\mathsf{true}}$ or $\underline{\mathsf{false}}$).

The next proposition characterizes universal solutions with respect to query answering, when the queries under consideration are unions of conjunctive queries. First, it

shows that *certain*$(q, I) = q(J)_{\downarrow}$ whenever $J$ is a universal solution and $q$ is a union of conjunctive queries. Concrete instances of this result in the LAV setting have been established in [2]. Another instance of this result has also been noted for the GAV setting with key/foreign key constraints in [5]. Thus, evaluation of conjunctive queries on an arbitrarily chosen universal solution gives precisely the set of certain answers. Moreover, the second statement of the proposition shows that the universal solutions are the only solutions that have this property. This is further justification for using universal solutions for data exchange.

**Proposition 2.** *Consider a data exchange setting with* $\mathbf{S}$ *as the source schema,* $\mathbf{T}$ *as the target schema, and such that the dependencies in the sets* $\Sigma_{st}$ *and* $\Sigma_t$ *are arbitrary.*

1. *Let* $q$ *be a union of conjunctive queries over the target schema* $\mathbf{T}$*. If* $I$ *is a source instance and* $J$ *is a universal solution, then certain*$(q, I) = q(J)_{\downarrow}$.
2. *Let* $I$ *be a source instance and* $J$ *be a solution such that for every conjunctive query* $q$ *over* $\mathbf{T}$*, we have that certain*$(q, I) = q(J)_{\downarrow}$*. Then* $J$ *is a universal solution.*

The following result follows from Corollary 1 and Part 1 of Proposition 2.

**Corollary 2.** *Assume a data exchange setting where* $\Sigma_{st}$ *is a set of tgds, and* $\Sigma_t$ *is the union of a weakly acyclic set of tgds with a set of egds. Let* $q$ *be a union of conjunctive queries. For every source instance* $I$*, the set certain*$(q, I)$ *can be computed in polynomial time in the size of* $I$.

The state of affairs changes dramatically when conjunctive queries with inequalities are considered. The next proposition shows that there is a simple Boolean conjunctive query $q$ with inequalities such that no universal solution can be used to obtain the certain answers of $q$ by evaluating $q$ on that universal solution. This proposition also shows that in this particular case, there is another conjunctive query $q^*$ with inequalities such that the certain answers of $q$ can be obtained by evaluating $q^*$ on the canonical universal solution.

**Proposition 3.** *Let* $S$ *be a binary source relation symbol,* $T$ *a binary target relation symbol,* $S(x, y) \rightarrow \exists z(T(x, z) \wedge T(z, y))$ *a source-to-target dependency, and* $q$ *the following Boolean conjunctive query with one inequality:* $\exists x \exists y(T(x, y) \wedge (x \neq y))$.

1. *There is a source instance* $I$ *such that certain*$(q, I) =$ $\underline{\mathsf{false}}$*, but* $q(J) =$ $\underline{\mathsf{true}}$ *for every universal solution* $J$.
2. *Let* $q^*$ *be the query* $\exists x \exists y \exists z(T(x, z) \wedge T(z, y) \wedge (x \neq y))$*. If* $I$ *is a source instance and* $J$ *is the canonical universal solution, then certain*$(q, I) = q^*(J)$.

In view of Proposition 3, we address next the question of whether, given a conjunctive query with inequalities, it is always possible to find a query (not necessarily the same) that computes the certain answers when evaluated on the canonical universal solution.

## 5 Query Answering: Complexity and Inexpressibility

It is known that in LAV data integration systems, computing the certain answers of conjunctive queries with inequalities is a coNP-hard problem [2]. It follows that in the data exchange setting, it is not possible to compute the certain answers of such queries

by evaluating them on the canonical universal solution or on any universal solution that is generated in polynomial time (unless P = NP). We take in Section 5.1 a closer look at conjunctive queries with inequalities. First we show (Theorem 3) that, in the data exchange setting, the problem of computing the certain answers for unions of conjunctive queries with inequalities is in coNP. Surprisingly, we show (Theorem 6) that there is a polynomial-time algorithm that computes the certain answers of unions of conjunctive queries with at most one inequality per disjunct. This is an optimal result because we also show (Theorem 5) that it is coNP-hard to compute the certain answers of unions of conjunctive queries with at most two inequalities per disjunct.

In the case of unions of conjunctive queries with at most one inequality per disjunct, the certain answers can be computed in polynomial time from an arbitrary universal solution. However, Section 5.2 shows (with no unproven complexity-theoretic assumptions such as P $\neq$ NP) that there is a conjunctive query $q$ with one inequality whose certain answers cannot be computed by rewriting $q$ to a first-order query $q^*$ and then evaluating $q^*$ on the canonical universal solution. We begin by formally introducing the decision problem associated with the computation of the set of certain answers.

**Definition 8.** Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

1. Let $q$ be a $k$-ary query over the target schema $\mathbf{T}$. *Computing the certain answers of* $q$ is the following decision problem: given a source instance $I$ over $\mathbf{S}$ and a $k$-tuple $t$ of constants from $I$, is it the case that $t \in \mathit{certain}(q, I)$?
2. Let $q$ be a Boolean query over the target schema $\mathbf{T}$. *Computing the certain answers of* $q$ is the following decision problem: given a source instance $I$ over $\mathbf{S}$, is it the case that $\mathit{certain}(q, I) = \underline{\mathsf{true}}$?
3. Let $\mathcal{C}$ be a complexity class and $\mathcal{Q}$ a class of queries over the target schema $\mathbf{T}$. We say that *computing the certain answers of queries in* $\mathcal{Q}$ *is in* $\mathcal{C}$ if for every query $q \in \mathcal{Q}$, computing the certain answers of $q$ is in $\mathcal{C}$. We say that *computing the certain answers of queries in* $\mathcal{Q}$ *is* $\mathcal{C}$*-complete* if it is in $\mathcal{C}$ and there is at least one query $q \in \mathcal{Q}$ such that computing the certain answers of $q$ is a $\mathcal{C}$-complete problem.

Thus, computing the certain answers of a $k$-ary query $q$ is a decision problem. One can also consider a related function problem: given a source instance $I$, find the set $\mathit{certain}(q, I)$. The latter problem has a polynomial-time reduction to the former, since there are polynomially many $k$-tuples from $I$ and so we can compute the set $\mathit{certain}(q, I)$ by going over each such $k$-tuple $t$ and deciding whether or not $t \in \mathit{certain}(q, I)$.

### 5.1 Computational Complexity

Since the complexity-theoretic lower bounds and inexpressibility results presented in the sequel hold for LAV data integration systems with sound views defined by conjunctive queries, we review the definition of this type of data integration system first. A LAV data integration system with sound views defined by conjunctive queries is a special case of a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which $\Sigma_t = \emptyset$ and each source-to-target dependency in $\Sigma_{st}$ is a tgd of the form $S_i(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, where $S_i$ is some relation symbol of the source schema $\mathbf{S}$ and $\psi_{\mathbf{T}}$ is an arbitrary conjunction of atomic formulas over the target schema $\mathbf{T}$. In what follows we will refer to such a setting as a LAV *setting*.

Abiteboul and Duschka [2] showed that in the LAV setting computing the certain answers of unions of conjunctive queries with inequalities is in coNP. We extend this by showing that the same upper bound holds in the data exchange setting, provided $\Sigma_{st}$ is a set of tgds and $\Sigma_t$ is a union of a set of egds with a weakly acyclic set of tgds.

**Theorem 3.** *Consider a data exchange setting in which $\Sigma_{st}$ is a set of tgds and $\Sigma_t$ is a union of a set of egds with a weakly acyclic set of tgds. Let $q$ be a union of conjunctive queries with inequalities. Then computing the certain answers of $q$ is in* coNP.

We first note that, in the particular case when all the tgds in $\Sigma_t$ are full, the theorem can be proved by using the "small model property" (essentially this argument was used in [2] for the LAV setting). However, for the more general case when the tgds in $\Sigma_t$ may have existentially quantified variables, the proof is more involved. It is based on an extension of the chase, called the *disjunctive chase*; see the full version [9] for details.

Theorem 3 yields an upper bound in a fairly general data exchange setting for the complexity of computing the certain answers of unions of conjunctive queries with inequalities. It turns out, as we discuss next, that this upper bound is tight, even in fairly restricted data exchange settings. Specifically, computing certain answers for such queries is coNP-complete. Therefore no polynomial algorithm exists for computing the certain answers when the input is a universal solution, unless $\mathrm{P} = \mathrm{NP}$.

Abiteboul and Duschka [2] showed that in the LAV setting, computing certain answers of conjunctive queries with inequalities is coNP-complete. They also sketched a proof which, if correct, would establish that this problem is coNP-complete even for conjunctive queries with a single inequality. Unfortunately, the reduction is erroneous. A correct reduction cannot be produced without increasing the number of inequalities, since here we show that in the LAV setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct. Still, the result of Abiteboul and Duschka [2] is correct; in fact, the unpublished full version [3] of that paper contains a proof to the effect that in the LAV setting, computing certain answers of Boolean conjunctive queries with six inequalities is coNP-complete. A different proof of the same result can be extracted by slightly modifying the proof of Theorem 3.2 in van der Meyden [22]. Thus, the next result provides a matching lower bound for the complexity of computing the certain answers of conjunctive queries with inequalities.

**Theorem 4.** [2,22] *In the* LAV *setting, computing the certain answers of Boolean conjunctive queries with six or more inequalities is* coNP*-complete.*

It is an interesting technical problem to determine the minimum number of inequalities needed to give rise to a coNP-complete problem in this setting.

*Conjecture 1.* In the LAV setting, computing the certain answers of Boolean conjunctive queries with two inequalities is coNP-complete.

We have not been able to settle this conjecture, but have succeeded in pinpointing the complexity of computing the certain answers of *unions* of Boolean conjunctive queries with at most two inequalities per disjunct.

**Theorem 5.** *In the* LAV *setting, computing the certain answers of unions of Boolean conjunctive queries with at most two inequalities per disjunct is* coNP*-complete. In fact,*

*this problem is* coNP*-complete even for the union of two queries the first of which is a conjunctive query and the second of which is a conjunctive query with two inequalities.*

For unions of conjunctive queries with inequalities, Theorem 5 delineates the boundary of intractability, because the next theorem asserts that computing certain answers of unions of conjunctive queries with at most one inequality per disjunct can be solved in polynomial time by an algorithm that runs on universal solutions.

**Theorem 6.** *Assume a data exchange setting in which $\Sigma_{st}$ is a set of tgds, and $\Sigma_t$ is the union of a weakly acyclic set of tgds with a set of egds. Let $q$ be a union of conjunctive queries with at most one inequality per disjunct. Let $I$ be a source instance and let $J$ be an arbitrary universal solution for $I$. Then there is a polynomial-time algorithm with input $J$ that computes certain$(q, I)$.*

**Corollary 3.** *Assume a data exchange setting in which $\Sigma_{st}$ is a set of tgds, and $\Sigma_t$ is the union of a weakly acyclic set of tgds with a set of egds. Let $q$ be a union of conjunctive queries with at most one inequality per disjunct. Then there is a polynomial-time algorithm for computing the certain answers of $q$.*

### 5.2 First-Order Inexpressibility

The following theorem shows that, in general, even for a conjunctive query $q$ with just one inequality, there is no first-order query $q^*$ that computes the certain answers when evaluated on the canonical universal solution. This is in strong contrast with the polynomial-time algorithm that we have seen earlier (Theorem 6). It is also in contrast with the second part of Proposition 3, where we have seen a particular example for which such a $q^*$ exists. The proof of the theorem combines Ehrenfeucht-Fraïssé games with the chase procedure.

**Theorem 7.** *There is a LAV setting with source $I$ and there is a Boolean conjunctive query $q$ with one inequality, for which there is no first-order query $q^*$ over the canonical universal solution $J$ such that certain$(q, I) = q^*(J)$.*

In the full version we also show that the result holds even if we allow the first-order formula $q^*$ to contain the predicate const that distinguishes between constants and nulls.

The next result, of particular interest to query answering in the data integration context, is a corollary to the proof of Theorem 7. It shows that for conjunctive queries with just one inequality we cannot in general find any first-order query over the *source* schema that, when evaluated on the *source* instance, computes the certain answers.

**Corollary 4.** *There is a LAV setting with source $I$ and there is a Boolean conjunctive query $q$ with one inequality, for which there is no first-order query $q^*$ over the source schema such that certain$(q, I) = q^*(I)$.*

## 6 Concluding Remarks

We plan to further investigate how universal solutions can be used for query answering in data exchange. We wish to characterize when a query $q$ can be rewritten to a first-order query $q^*$ such that the certain answers of $q$ can be computed by evaluating $q^*$ on a

universal solution. We also wish to understand how well the certain answers of a query can be approximated by evaluating the same query on a universal solution and how this differs from universal solution to universal solution. Finally, an important direction is extending the notion of universal solution to cover data exchange between nested (e.g. XML) schemas.

## References


1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and Translation for Heterogeneous Data. In *ICDT*, pages 351–363, 1997.
2. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.
3. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. Unpublished full version of [2], 2000.
4. C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *JACM*, 31(4):718–741, 1984.
5. A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data Integration under Integrity Constraints. In *CAiSE*, pages 262–279, 2002.
6. M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion Dependencies and their Interaction with Functional Dependencies. *JCSS*, 28(1):29–59, 1984.
7. S. S. Cosmadakis and P. C. Kanellakis. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*, volume 3, pages 163–184. 1986.
8. R. Fagin. Horn Clauses and Database Dependencies. *JACM*, 29(4):952–985, Oct. 1982.
9. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. IBM Research Report, Nov. 2002.
10. M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *AAAI*, pages 67–73, 1999.
11. A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.
12. R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*, pages 455–468, 1990.
13. M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.
14. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, May 1995.
15. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, Dec. 1979.
16. D. Maier, J. D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM TODS*, 9(2):283–308, June 1984.
17. J. A. Makowsky. Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples. *JCSS*, 34(2/3):266–292, April/June 1987.
18. R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.
19. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
20. N. C. Shu, B. C. Housel, and V. Y. Lum. CONVERT: A High Level Translation Definition Language for Data Conversion. *Communications of the ACM*, 18(10):557–567, 1975.
21. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *TODS*, 2(2):134–174, 1977.
22. R. van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *JCSS*, 54:113–135, 1997.
23. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

# Reformulation of XML Queries and Constraints

Alin Deutsch[1] and Val Tannen[2]

[1] UC San Diego, deutsch@cs.ucsd.edu
[2] University of Pennsylvania, val@cis.upenn.edu

**Abstract.** We state and solve the query reformulation problem for XML publishing in a general setting that allows mixed (XML and relational) storage for the proprietary data and exploits redundancies (materialized views, indexes and caches) to enhance performance. The correspondence between published and proprietary schemas is specified by views in both directions, and the same algorithm performs rewriting-with-views, composition-with-views, or the combined effect of both, unifying the Global-As-View and Local-As-View approaches to data integration. We prove a completeness theorem which guarantees that under certain conditions, our algorithm will find a minimal reformulation if one exists. Moreover, we identify conditions when this algorithm achieves optimal complexity bounds. We solve the reformulation problem for constraints by exploiting a reduction to the problem of query reformulation.

## 1 Introduction

The problem of **query reformulation** is a very general one: given two schemas $P$ and $S$ and a correspondence $CR$ between them, and given a query $Q$ formulated in terms of $P$, find a query $X$ formulated in terms of $S$ that is equivalent to $Q$ modulo the correspondence $CR$. Reformulation algorithms have many uses in database technology, for example in data integration where $P$ is the *global* integrated schema and $S$ gathers the *local* schemas of the actual data sources, or in schema evolution where $P$ is the old schema and $S$ is the new schema.

In this paper our motivation and specific challenges come from **XML publishing**, where $P$ is the *public* XML schema and $S$ is the *storage* schema of the proprietary data from which selected portions are published. Typically, the proprietary data resides in relational databases (RDB) and lately also in native XML document storage (e.g., if acquired through XML exchange). Clients formulate queries against the public XML schema (in our case in XQuery [33]) and the publishing system must *reformulate* these into queries on the storage schema data in order to answer them.

A central problem is how to model the schema correspondence $CR$. Data integration systems use one of two approaches for the analogous problem [20, 22]: "Global-As-View" (GAV) and "Local-As-View" (LAV) with the views themselves (sometimes called *mappings*) expressed in a query language. We shall use these acronyms but keep in mind that for us

GAV views : storage $\longrightarrow$ public  LAV views : public $\longrightarrow$ storage

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 225–241, 2003.

In fact, neither of these two approaches used in *isolation* is flexible enough for our problem. The GAV approach is convenient for **hiding portions of the proprietary data**: the view definition can simply project/select them away. This cannot be done in a LAV approach, since the view's input is in this case the published data, from which the hidden information is missing. On the other hand, we will also want to tune the performance of the publishing system by, e.g., caching query results or redundantly storing some of the native XML data in relational databases in order to exploit the more mature relational technology. The resulting **redundancies in the stored data** can be easily exploited in the LAV approach and will typically lead to multiple reformulations.[1] However, existing techniques for the GAV approach do not handle such redundancies properly (see related work). We conclude that in common XML publishing scenarios we need schema correspondences specified using a *combination* of both kind of views (GLAV), each of them a mapping from a portion of the storage schema to a portion of the public schema, or conversely. To facilitate design and administration tasks, agreeing with [8] that XML encodings of relational schemas are easily understood and used, we shall assume that these views are expressed in XQuery.

Finally, we consider integrity constraints on both the public and the storage schema. (Note that the presence of constraints will never reduce but may often *expand* the space of possible reformulations.) While much is known about relational constraints, XML constraint formalisms are still "under construction". We follow here our proposal [12] for a class of **X**ML **I**ntegrity **C**onstraints (`XIC`) whose expressive power captures a considerable part of XML Schema [32,4] including keys and "keyrefs" and quite a bit beyond.

Therefore, in this paper we study the following problem:

**Given:**
– the public schema $P$ as XML, with constraints
– the storage schema $S$: mixed, RDB + XML, with constraints
– the client query $Q$ formulated over $P$ in XQuery
– the schema correspondence $CR$ between $P$ and $S$ formulated as
  – a (simple) encoding of RDB into XML
  – mappings(views) between portions of $P$ and $S$ in both directions (GLAV)

**Find:**
– one or more queries $X$ formulated over $S$, such that
– $X$ is equivalent to $Q$ under $CR$

Our **approach** to query reformulation is to "compile" the XML reformulation problem into a relational reformulation problem and then use an algorithm that we have proposed earlier together with Lucian Popa [10]. The different ingredients of this strategy are sketched in the following steps 1–5.

[1] When our reformulation algorithm produces multiple candidates, these should be further compared using application-specific cost models. This important step is outside the scope of this paper (but see [27]).

**Step 1.** We encode the stored relational schemas into XML (pick one of several straightforward encodings). Then, the DB administrator can define mappings RDB→XML or XML→RDB just by writing them in XQuery. Thus, the schema correspondence is given by several **XQuery views** (in both directions). We also take integrity constraints into consideration, on the relational part as *disjunctive embedded dependencies* (DEDs), see [1,14] and section 3, and on the XML part as `XIC`s, see section 4.2 and [12].[2]

**Step 2.** Like [8,25] we follow [16] in splitting **XQuery = navigation part + tagging template** corresponding to the two phases in the operational semantics of XQuery [33], see section 4.2. Previous research has addressed the efficient implementation of the second phase [30,15]. Only the first phase depends on the schema correspondence so we focus on **reformulating the navigation part of XQueries**.

**Step 3.** We define a **generic relational encoding for XML**[3] whose schema we call $\mathcal{X}$ (see section 4.1). Then, the XML encoding (see Step 1) of the stored relational schema is captured by a set of DEDs relating these schemas to schema $\mathcal{X}$, as explained in section 4.3.

**Step 4.** We define a syntactic restriction of XQuery, the *behaved* queries, that are still very powerful (see section 4). We give algorithms that **translate**: (1) the navigation part (see Step 2) of a behaved XQuery into a relational union of conjunctive queries over $\mathcal{X}$, call it $B$, (2) the behaved XQuery views in the schema correspondence (see Step 1) into sets of relational DEDs over $\mathcal{X}$ (see Step 3), and (3) the `XIC`s from both schemas (see Step 1) also into sets of relational DEDs over $\mathcal{X}$.

**Step 5.** We now have a relational query $B$ (see Step 5) that needs to be reformulated modulo equivalence under the set of all relational constraints coming from Steps 1, 3, and 4. For this we use the **C&B algorithm** [10]. We prove new theorems that show that our algorithm is indeed **complete** in that it finds all "minimal" reformulations (see sections 3 and 4.4).

Why "minimal"? Note that in general a query has infinitely many reformulations just by trivially adding repeating scans (eg., items in the `from` clause). We call a reformulation **minimal** (see section 3) if it performs a minimal number of scans over source data, in the sense that we cannot remove a scan without compromising equivalence to the original query. Note also that if a query has any reformulation then it will have a minimal one as well.

Our approach is summarized in Figure 1 which happens to also be describing the architecture of the MARS (mixed and redundant storage) system that implements it (more in section 6).

The **constraint reformulation** problem also arises naturally in the XML publishing scenario. If a certain constraint $d$ on the published data is desired, an

[2] Because of the encoding of RDB in XML we can also use `XIC`s for constraints between the RDB and XML parts.

[3] Interestingly, in a mixed RDB + XML situation we encode RDB in XML to make view and query specification user-friendly, but then we encode XML in RDB for the automated query processing!

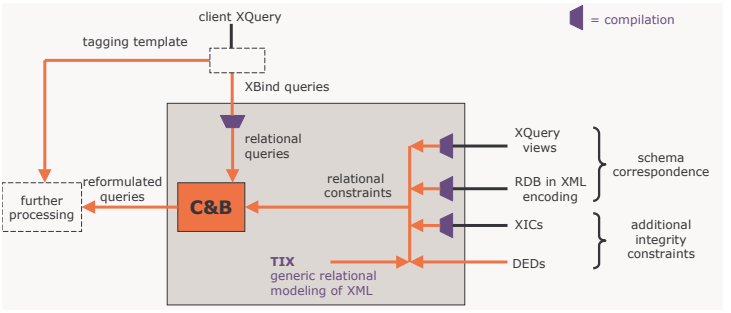



**Fig. 1.** MARS architecture

administrator may be able to achieve this by enforcing *additional* constraints on the storage data. But which ones? We could guess, and test as follows: compile the schema correspondence and the storage constraints into a set $\Delta$ of relational constraints; compile the desired XML constraints on the published schema into a set $D$ of relational constraints; then ask if $\Delta \models D$, using the chase to test it. But there may be a better way. Namely, *reformulate* $d$ into a storage constraint $\delta$ that is equivalent to $d$ modulo the schema correspondence. It may then be easier to redesign the storage constraints in order to enforce $\delta$ (hence $d$). In section 5 we describe one approach to such reformulation.

## 2 Contributions and Related Work

The conceptual contribution of this work to the XML publishing research topic is a uniform solution to the problem of finding **minimal reformulations of XQueries**, when the schema correspondence is given by a combination of GAV- and LAV-style XQuery views. Our solution allows mixed storage (RDB and XML), and integrity constraints on both the public and storage schemas. Our approach unifies the LAV and GAV data integration scenarios by achieving the combined effect of rewriting-with-views, composition-with-views and minimization. Moreover, we show how to **apply our algorithm to constraint reformulation** by exploiting the inter-reducibility of the problems of query containment and dependency implication (section 5). All of this is made possible by the following technical contributions.

We **reduce** this XML problem to a similar problem involving only relational queries and relational dependencies. We give translation algorithms for this reduction (see step 5 in section 1). We prove a **relative completeness theorem** for the translation (Theorem 2) that says in essence that any existing solution of the XML problem can be recovered from some minimal relational reformulation that is a solution of the relational translated problem.

The translated problem consist of finding minimal reformulations of unions of conjunctive queries under sets of disjunctive embedded dependencies (section 3).

We solve this problem with the C&B algorithm. This algorithm was introduced in [10] and extended in [14] to also deal with unions and disjunctions. A limited completeness theorem was shown in [10], for the case when the constraints correspond to just LAV views, no views in the reverse direction and no additional constraints on the schemas. In this paper we prove a much more general **C&B completeness theorem**, namely for any set of constraints that yield a terminating chase (Theorem 1). By combining Theorem 2 with Theorem 1, we conclude that our solution to the XML reformulation problem is **overall complete** (Corollary 1). Our completeness results hold only for the *behaved* queries (defined in section 4), and for *bounded* XML constraints (in section 4.4). In fact, the method is applicable to larger classes of queries, views and constraints, as long as we can compile them and apply the chase, being understood that we don't have completeness guarantees anymore. From a practical perspective, we argue that the features that we cover are in our experience the most common ones anyway.

The limitations of the method are not arbitrary. To **calibrate our results** we first show that checking minimality under dependencies is as hard as deciding query containment (Proposition 2). This allows us to use lower bounds from [12] on the containment of the navigation part of XQueries to show that the restrictions we have imposed are quite essential. Indeed, we conclude that even modest extensions of the class of behaved XQueries will make our algorithm **incomplete** (unless $NP = \Pi_2^p$). We also conclude that even modest use of *unbounded* XML constraints makes the overall problem **undecidable**.

**Related work.** For XML publishing in the pure GAV approach, and when the storage schema is purely relational, our system subsumes the expressive power of XPeranto [30] and SilkRoute [16]. For the LAV approach, we handle as particular instances XML publishing as in Agora [25] and STORED [9] and purely relational integration as in the Information Manifold [24]. The problem of rewriting regular path queries with inverse (RPQIs) with RPQI views in a LAV semistructured data context was addressed in [6,7]. [26] gives a complete algorithm for rewriting semistructured queries with semistructured views. However, the main technical difficulties we have solved for the translation (see above) are in XQuery but not in RPQIs or the semistructured queries from [26]. While Agora captures implicitly some of the constraints inherent in the relational encoding of XML, none of the above approaches allow for *additional* constraints on the schemas. [18] and [31] propose algorithms which do take into account some constraints (e.g., referential integrity constraints) and they run in PTIME. The disadvantage here is missing rewritings (unless P=NP, since the problem is NP-hard). [17] reduces the schema correspondence given in the combined LAV and GAV approaches (GLAV) to a pure GAV correspondence. The technique does not apply to our publishing scenario because the obtained reformulation accesses *all* sources containing relevant information and thus defeats the purpose of redundant storage. [5] extends the ideas in [17] to allow for a restricted class of constraints on the published schema. [7] solves the problem for RPQI queries and RPQI views. According to our new completeness result, the C&B is a complete

algorithm for minimization of (unions of) conjunctive queries under disjunctive embedded dependencies. The early work on query minimization (see [1]) did not handle dependencies. [2] lists as an open problem even the special case of the minimization of an SPJ query under functional dependencies. [19] minimizes conjunctive queries under inclusion dependencies. All of these (and more general cases) are solved by the C&B algorithm.

## 3 Relational Query Reformulation: The C&B Algorithm

**Review: Capturing views with dependencies.** The key observation that enables the uniform treatment of views and integrity constraints by the C&B algorithm is the fact that conjunctive query views can be captured by *embedded dependencies* [1] relating the input of the defining query with its output. For example, consider the view defined by

$$V(x,z) \leftarrow A(x,y), B(y,z)$$

In any instance over the schema $\{A, B, V\}$, the extent of relation $V$ coincides with the result of this query if and only if the following dependencies hold:

$$(c_V) \quad \forall x \forall y \forall z \; [A(x,y) \wedge B(y,z) \rightarrow V(x,z)]$$
$$(b_V) \quad \forall x \forall z \; [V(x,z) \rightarrow \exists y \; A(x,y) \wedge B(y,z)]$$

$(c_V)$ states the inclusion of the result of the defining query in the extent of relation $V$, $(b_V)$ states the opposite inclusion.

**Review of C&B.** Assume that in addition, the following dependency holds on the database (it is an inclusion dependency):

$$(ind) \quad \forall x \forall y \; [A(x,y) \rightarrow \exists z \; B(y,z)]$$

Suppose that we want to reformulate the query

$$Q(x) \leftarrow A(x,y)$$

First, the query is *chased* with all available dependencies, until no more chase steps apply (see [1] for a detailed definition of the chase). The resulting query is called the *universal plan*. In our example, a chase step with $(ind)$ yields $Q_1$ below, which in turn chases with $(c_V)$ to the universal plan $Q_2$:

$$Q_1(x) \leftarrow A(x,y), B(y,z)$$
$$Q_2(x) \leftarrow A(x,y), B(y,z), V(x,z)$$

Notice how the chase step with $(c_V)$ brings the view into the chase result, and how this was only possible after the chase with the semantic constraint $(ind)$.

In the second phase of the algorithm (called the *backchase*) the *subqueries* of the universal plan are inspected and checked for equivalence with $Q$. Subqueries are obtained by retaining only a subset of the atoms in the body of the universal

plan, using the same variables in the head. For example, $S(x) \leftarrow V(x,z)$ is a subquery of $Q_2$ which turns out to be equivalent to $Q$ under the available constraints, as can be checked by chasing $S$ "back" to $Q_2$ using $(b_V)$.

**A New Completeness Result.** It is not accidental that we discovered a reformulation among the subqueries of the universal plan; in fact, in theorem 1 we give a theoretical guarantee that *all minimal* reformulations can be found this way. We say that a query $R$ is *minimal under a set of constraints* $C$ (or $C$-minimal) if no relational atoms can be removed from $R$'s body, even after adding arbitrarily many equality atoms, without compromising the equivalence to $R$ under $C$. Recalling the example in section 3, $T(x) \leftarrow A(x,y), V(x,z)$ is not minimal under the constraints $\{(c_V), (b_V), (ind)\}$, because we can remove the $A$-atom (without adding equalities) to obtain $M(x) \leftarrow V(x,z)$, which is equivalent to $T$, as can be checked by chasing. A query $R$ is a *minimal reformulation* of query $Q$ under $C$ if it is $C$-minimal and equivalent to $Q$ under $C$ ($C$-equivalent to $Q$).

**Theorem 1.** *Let $Q$ be a conjunctive query and $D$ be a set of embedded dependencies. Assume that there is some terminating chase sequence of $Q$ with $D$, yielding the universal plan $U$. Then any minimal reformulation of $Q$ under $D$ is isomorphic to a subquery of $U$.*

This result adds significant value to the one in [10], where we showed the completeness of the C&B when only views are allowed (i.e. we allow the constraints capturing the views such as $(c_V), (b_V)$, but no additional integrity constraints such as $(ind)$). Of course, checking the existence of a terminating chase sequence for a conjunctive query and arbitrary embedded dependencies is undecidable. In [10], we show that all chase sequences terminate when only the dependencies capturing the views are used. For the case of additional dependencies, we identify here a property that guarantees the termination of any chase sequence for any query.

**Set of constraints with stratified-witness.** Given a set $C$ of constraints, define its *chase flow graph* $G = (V, E)$, as a directed graph whose edge labels can be either $\forall$ or $\exists$. $G$ is constructed as follows: for every relation $R$ of arity $a$ mentioned in $C$, $V$ contains a node $R^i$ $(1 \leq i \leq a)$. For every pair of relations $R, R'$ of arities $a, a'$ and every constraint $\forall \mathbf{x}\, [\ldots \wedge R(u_1, \ldots, u_a) \wedge \ldots \rightarrow R'(v_1, \ldots v_{a'}) \ldots]$ in $C$, $E$ contains the edges $(R_i, R'_j)_{1 \leq i \leq a, 1 \leq j \leq a'}$. Also, whenever the equality $x = y$ appears in the conclusion of the implication, and $x, y$ appear as the $i, j$-th component of $R$, resp. $R'$, $E$ contains the edge $(R_i, R'_j)$. Moreover, if for some $j$ the variable $v_j$ is existentially quantified, the edges $(R_i, R'_j)_{1 \leq i \leq a}$ are labeled with $\exists$, otherwise they are labeled with $\forall$. We say that a set of constraints has *stratified-witness* if it has no cycles through $\exists$-edges.

Denoting with $|Q|$ the size of query $Q$, with $a$ the maximum arity of a relation in the schema and with $l$ the maximum number of $\exists$-edges on a path in the chase flow graph, we have the following

**Proposition 1 (with Lucian Popa).** *The chase of any query $Q$ with any set of constraints with stratified-witness terminates, and the size of the resulting query is in $O(|Q|^{a^{l+1}})$.*

This condition is efficiently checkable, and it subsumes known guarantees of the chase termination for various classes of dependencies: functional dependencies, total/full dependencies, typed 1-non-total dependencies, typed dependencies with identical sets of total attributes [3].[4]

*Remarks. 1.* Notice that any pair of inclusion dependencies used to capture a view (recall $(c_V), (b_V)$ from page 230) violates the stratified-witness condition. However, the chase is guaranteed to terminate nevertheless, using the additional key observation that the introduction of the view symbol $V$ by a chase step with $(c_V)$ can never trigger a chase step with $(b_V)$. This effectively breaks the $\exists$-cycle appearing in the chase flow graph.

*2.* When the C&B is used in the following particular scenario: (i) $Q$ is posed against the public schema $P$, (ii) $D$ gives the correspondence between $P$ and storage schema $S$, and (iii) in the backchase phase we consider only subqueries expressed solely in terms of $S$, we obtain a complete algorithm for finding minimal reformulations.

*3.* By theorem 1, the C&B algorithm is a complete procedure for minimization of conjunctive queries under stratified-witness dependencies, generalizing existing procedures (see related work).

**Calibrating the result.** Since the backchase checks subqueries for equivalence under dependencies to the universal plan, the C&B algorithm inherits the complexity lower bounds of the equivalence check. Moreover, the C&B cannot be complete if equivalence is undecidable. A natural question is whether there are alternate algorithms that do better (are complete even when equivalence is not decidable, and have lower complexity when it is). The answer is no:

**Proposition 2.** *The problem of deciding minimality of a conjunctive query over all models that belong to some class $C$ and satisfy a set of dependencies is at least as hard as deciding containment of conjunctive queries over the class $C$.*

In particular, the class $C$ may be specified as all models satisfying a set of dependencies. Undecidability of containment under dependencies therefore implies that the set of minimal reformulations under dependencies is not recursive.

It turns out that the C&B algorithm is asymptotically optimal even when used as an alternative to classical algorithms for rewriting with views in the absence of additional integrity constraints (such as Minicon [28]): the associated decision problem is checking the existence of a rewriting using solely the views, in the absence of constraints. The C&B-based solution would consist in picking from the universal plan $U$ the maximal subquery that mentions only views, and

[4] The chase flow graph is similar to the graph used to determine the existence of stratified normal forms for ILOG programs [21]. These invent object identities, just like the chase invents new variables.

checking its equivalence to $U$. The complexity analysis reveals that the resulting algorithm is in NP in the size of the query, which is optimal according to [23].

**Extension: DEDs.** The theorem holds even when $Q$ is a union of conjunctive queries and $D$ is a set of *disjunctive embedded dependencies* (DEDs), as introduced in [14], which extended the chase to DEDs. Their general form is

$$\forall \mathbf{x}\ [\phi(\mathbf{x}) \rightarrow \bigvee_{i=1}^{l} \exists \mathbf{z_i}\ \psi_i(\mathbf{x}, \mathbf{z_i})] \tag{1}$$

where $\mathbf{x}, \mathbf{z_i}$ are tuples of variables and $\phi, \psi_i$ are conjunctions of *relational atoms* of the form $R(w_1, \ldots, w_l)$ and *(in)equality atoms* of the form $(w \neq w')\ w = w'$, where $w_1, \ldots, w_l, w, w'$ are variables or constants. $\phi$ may be the empty conjunction. We call such dependencies *disjunctive embedded dependencies (DEDs)*, because they contain the classical embedded dependencies [1] when $l = 1$. A proper DED is (`choice`) from `TIX`.

# 4 XML Query Reformulation

## 4.1 Using Relational Constraints to Capture XML

We treat mixed XML+relational storage uniformly by *reduction to a relational framework*. More specifically, following [12], we shall represent XML documents as relational instances[5] over the schema

$$\mathcal{X} = [\texttt{root}, \texttt{el}, \texttt{child}, \texttt{desc}, \texttt{tag}, \texttt{attr}, \texttt{id}, \texttt{text}].$$

The "intended meaning" of the relations in $\mathcal{X}$ reflects the fact that XML data is a tagged tree. The unary predicate `root` denotes the root element of the XML document, and the unary relation `el` is the set of all elements. `child` and `desc` are subsets of `el` $\times$ `el` and they say that their second component is a child, respectively a descendant of the first component. `tag` $\subseteq$ `el` $\times$ *string* associates the tag in the second component to the element in the first. `attr` $\subseteq$ `el` $\times$ *string* $\times$ *string* gives the element, attribute name and attribute value in its first, second, respectively third component. `id` $\subseteq$ *string* $\times$ `el` associates the element in the second component to a string attribute in the first that uniquely identifies it (if DTD-specified ID-type attributes exist, their values can be used for this). `text` $\subseteq$ `el` $\times$ *string* associates to the element in its first component the string in its second component.

*Relational translation of XML tree navigation.* Consider an XPath expression $q$ defined as $//a$, which returns the set of nodes reachable by navigating to a descendant of the root and from there to a child tagged "a". Assume also that we materialize the view $v$ defined as $//.//a$, i.e. which contains all "a"-children

[5] We emphasize that this does not mean that the XML data is necessarily stored according to the relational schema $\mathcal{X}$. Regardless of its physical storage, we reason about XML data using $\mathcal{X}$ as its virtual relational view.

of descendants of descendants of the root. We can translate $q, v$ as conjunctive queries $Q, V$ over schema $\mathcal{X}$ (see [12] for details):

$$Q(y) \leftarrow \texttt{root}(r), \texttt{desc}(r, x), \texttt{child}(x, y), \texttt{tag}(y, '' a'')$$
$$V(y) \leftarrow \texttt{root}(r), \texttt{desc}(r, u), \texttt{desc}(u, x), \texttt{child}(x, y), \texttt{tag}(y, '' a'')$$

Clearly, under arbitrary interpretations of the `desc` relation, the two are not equivalent, and $Q$ cannot be reformulated to use $V$. But on intended interpretations, the `desc` *relation is transitive* and therefore $R(d) \leftarrow V(d)$ is a reformulation for $Q$ using $V$. Any reformulation algorithm must take into account such constraints as transitivity on the intended models of $\mathcal{X}$ lest it should miss basic reformulations.

`TIX`*: Constraints inherent in the XML data model.* Some (but not all!) of the intended meaning of signature $\mathcal{X}$ is captured by the set `TIX` (**T**rue **I**n **X**ML) of relational constraints (only the most interesting ones are listed): [6]

(base) $\forall x, y\ [\ \texttt{child}(x, y) \rightarrow \texttt{desc}(x, y)\ ]$

(trans) $\forall x, y, z\ [\ \texttt{desc}(x, y) \wedge \texttt{desc}(y, z) \rightarrow \texttt{desc}(x, z)\ ]$

(refl) $\forall x\ [\ \texttt{el}(x) \rightarrow \texttt{desc}(x, x)\ ]$

(someTag) $\forall x\ [\ \texttt{el}(x) \rightarrow \exists t\ \texttt{tag}(x, t)\ ]$

(oneTag) $\forall x, t_1, t_2\ [\ \texttt{tag}(x, t_1) \wedge \texttt{tag}(x, t_2) \rightarrow t_1 = t_2\ ]$

(keyId) $\forall s, e_1, e_2\ [\ \texttt{id}(s, e_1) \wedge \texttt{id}(s, e_2) \rightarrow e_1 = e_2\ ]$

(oneAttr) $\forall x, n, v_1, v_2\ [\ \texttt{attr}(x, n, v_1) \wedge \texttt{attr}(x, n, v_2) \rightarrow v_1 = v_2\ ]$

(noLoop) $\forall x, y\ [\ \texttt{desc}(x, y) \wedge \texttt{desc}(y, x) \rightarrow x = y\ ]$

(oneParent) $\forall x, y, z\ [\ \texttt{child}(x, z) \wedge \texttt{child}(y, z) \rightarrow x = y\ ]$

(oneRoot) $\forall x, y\ [\ \texttt{root}(x) \wedge \texttt{root}(y) \rightarrow x = y]$

(topRoot) $\forall x, y\ [\ \texttt{desc}(x, y) \wedge \texttt{root}(y) \rightarrow \texttt{root}(x)]$

(`line`) $\forall x, y, u\ [\ \texttt{desc}(x, u) \wedge \texttt{desc}(y, u) \rightarrow x = y \vee \texttt{desc}(x, y) \vee \texttt{desc}(y, x)]$

(choice) $\forall x, y, z\ [\ \texttt{child}(x, y) \wedge \texttt{desc}(x, z) \wedge \texttt{desc}(z, y) \rightarrow x = z \vee y = z]$

Note that these axioms are First-Order incomplete; they don't even prove $\forall x \forall y\ \texttt{desc}(x, y) \rightarrow x = y \vee \exists z\ \texttt{child}(x, z) \wedge \texttt{desc}(z, y)$. Still they are special because they are sufficient to give an optimal, chase-based decision procedure for containment of XQueries from the fragment with NP-complete containment [12].

Notice that except for (`line`) and (`choice`), all constraints in `TIX` are *embedded dependencies* (as [1] calls them, but also known as tuple- and equality-generating dependencies [3]) for which a deep and rich theory has been developed. (`line`) contains disjunction but so do XQueries. Extending the theory to *disjunctive embedded dependencies* is fairly straightforward [14].

[6] A collection $D_1, \ldots, D_n$ of XML documents is represented by the disjoint union of schemas $\mathcal{X}_i$ and the union of constraints in each $\texttt{TIX}_i$, where each $\mathcal{X}_i$ ($\texttt{TIX}_i$) is obtained from $\mathcal{X}$ (resp. `TIX`) by subscripting all relational symbols with $i$.

*Transitive Closure and Treeness.* Observe that `(base), (trans), (refl)` above only guarantee that `desc` *contains* its intended interpretation, namely the reflexive, transitive closure of the `child` relation. There are many models satisfying these constraints, in which `desc` is interpreted as a proper superset of its intended interpretation, and it is well-known that we have no way of ruling them out using first-order constraints, because transitive closure is not first-order definable. Similarly, the "treeness" property of the `child` relation cannot be captured in first-order logic. The fact that we can nevertheless decide equivalence of behaved XQueries (containing descendant navigation) over the intended interpretation using the constraints in `TIX` and classical relational (hence first-order) techniques comes therefore as a pleasant surprise.

### 4.2 XML Queries and Constraints

According to their operational semantics, XQueries compute in two phases. First, the navigation part of an XQuery searches the input XML tree(s) binding the query variables to nodes or string values. In a second phase that uses the tagging template a new element of the output tree is created for each tuple of bindings produced in the first phase (see example 1 below).

**Describing the navigational part: decorrelated XBind queries.** In this paper, we focus on reformulating the navigational part of a client XQuery. In order to describe it, we introduce a simplified syntax that disregards the element construction, focusing only on binding variables and returning them. We call the queries in this syntax XBind queries. Their general form is akin to conjunctive queries. Their head returns a tuple of variables, and the body atoms can be purely relational or are predicates defined by XPath expressions with restrictions (see [12,13] for their syntax). The predicates can be binary, of the form $[p](x, y)$, being satisfied whenever $y$ belongs to the set of nodes reachable from node $x$ along the path $p$. Alternatively, predicates are unary, of form $[p](y)$, if $p$ is an absolute path starting at the root.

*Example 1.* Consider the query $Q$ defined as

```
for $a in distinct(//author/text()) return
    <item> <writer>$a</writer>
           {for   $b in //book, $a1 in $b/author/text(), $t in $b/title
           where  $a = $a1 return $t}
    </item>
```

Note the nested query shown in braces, which is correlated with the outer one through free variable $a. It returns *copies* of the `title` subelements of books whose author $a1 coincides with $a. The direct, nested loop-based evaluation of $Q$ is inefficient. Research in evaluating correlated SQL queries suggests an alternative strategy that consists in breaking the query into two decorrelated queries which can be efficiently evaluated separately and then putting together their results using an outer join [29]. We will borrow this technique, obtaining for $Q$ the two decorrelated XBind queries below ($ signs are dropped from the variable names). $Xb_o$ ($Xb_i$) computes the bindings for the variables introduced

in the outer (inner) `for` loop. Notice that $Xb_i$ also outputs the value of free variable $a in order to preserve the correlation between bindings.

$$Xb_o(a) \leftarrow [//author/text()](a)$$
$$Xb_i(a, b, a1, t) \leftarrow Xb_o(a), [//book](b), [./author/text()](b, a1), [./title](b, t), a = a1$$

In summary, we describe the navigational part of an XQuery by a set of decorrelated XBind queries. Using the translation of XPath expressions to conjunctions of relational atoms from signature $\mathcal{X}$ (sketched on page 234 and detailed in [12]), we obtain a straightforward translation of any XBind query to a union of conjunctive queries.

**XML Integrity Constraints (`XIC`s).** In [12], we use the same syntax for predicates defined by XPath expressions to define a class of XML integrity constraints (`XIC`s). It turns out that `XIC`s are related to XBind queries in the same way in which embedded dependencies are related to conjunctive queries: implication and containment are inter-reducible (Proposition 3). `XIC`s have the same general form as (1), but the relational atoms are replaced by predicates defined by restricted XPath expressions, just like for XBind queries (see (2) below). Using the same translation of XPath expressions as for XBind queries, we get a straightforward translation of `XIC`s to DEDs.

$$\forall x, y\ [/pers](x) \wedge [.//dog](x, y) \rightarrow \exists z\ [./pets](x, z) \wedge [.//](z, y) \tag{2}$$

**Behaved XQueries.** The strategy of our algorithm is to compile each XBind query to a union of conjunctive queries, compile all XQuery views (not just their XBind parts; tagging part as well!) to DEDs and apply the C&Bs algorithm to the relational problem. There are of course XQuery features we cannot compile to dependencies. User-defined functions, aggregates and universally quantified path qualifiers [33] are the main examples. We emphasize that the soundness of the algorithm presented below holds for any query that is compilable relationally. However, its completeness is guaranteed only for a restricted class of XQueries, which we call behaved. In addition to ruling the non-compilable features out, behaved XQueries satisfy a few more restrictions.[7] The main restriction rules out navigation to parent and wildcard child (i.e. child of unspecified tag) (more on this counterintuitive restriction shortly). This class is still quite expressive: it allows navigation to ancestor,descendant and child of *specified* tag; disjunction and path alternation; inequalities; equalities on values (text and attributes) and on node identities. The query in example 1 is behaved. From a practical perspective, the features that we cover are in our experience the most common ones anyway, with the exception of aggregates. As discussed shortly, even modest relaxation of these restrictions results in incompleteness of reformulation, suggesting that different techniques are needed beyond this class of XQueries.

[7] See [13,11] for the detailed description of this class of XQueries, or [12] for the behaved fragment of XPath used in behaved XQueries.

### 4.3 Compiling Schema Correspondences

*Obstacles in capturing XQuery views with dependencies.* In section 3, we show how we express a conjunctive query view using two inclusion dependencies. This technique does not apply directly to XQuery views, which are more expressive: (i) XQueries contain nested, correlated subqueries in the return clause, (ii) they create new nodes, which do not exist in the input document, so there is no inclusion relationship between input and output node id sets, and (iii) they return deep, recursive copies of elements from the input. We sketch the solution using example 1 (see [11,13] for more details).

*Nested, Correlated Subqueries.* Recall that the navigation part of an XQuery is described by a set of decorrelated XBind queries ($Xb_o$,$Xb_i$ in example 1). Also recall that every XBind query can be straightforwardly translated to a union of conjunctive queries over schema $\mathcal{X}$. This union can now be captured with two DEDs, as shown on page 230.

*Construction of New Elements.* For every binding for $a, a new `item` element node is created whose identity does not exist anywhere in the input document, but rather is an invented value. Distinct bindings of $a result in distinct invented `item` elements. In other words, the identities of the `item` element nodes are the image of the bindings for $a under some injective function $F_{item}$.[8] We capture this function by extending the schema with the relational symbol $G_{item}$, intended as the graph of $F_{item}$ ($G_{item}(x, y) \Leftrightarrow y = F_{item}(x)$) and use dependencies to enforce this intended meaning.

*Deep Copies of Elements.* Here is how we capture the fact that $Q$ returns, for every binding of $a, a copy of the tree rooted at the `title`-element node which $t was bound to. We model copying by an injective function $F_t^a$ which, for a fixed $a, takes as argument any node $n$ in the tree rooted at $t, and outputs an invented node $n'$ that is a copy of $n$. We say that $n'$ is an ($a,$t)-copy of $n$ to emphasize that there is one copy of the tree rooted at $t for each value of $a. We represent the family of ($a,$t)-copy functions $\{F_t^a\}_{a,t}$ by the relation $C$: $\forall a \forall t\ F_t^a(n, n') \Leftrightarrow C(a, t, n, n')$. Again, we capture the intended meaning for $C$ using DEDs. The sample DED (3) states that if $n'$ is an ($a,$t)-copy of $n$, then the descendants of $n$ are ($a,$t)-copied as descendants of $n'$ ($Q$'s output is encoded as an instance over schema $\mathcal{X}_2$):

$$\forall a \forall t \forall n \forall n' \forall d\ [C(a, t, n, n') \wedge \mathtt{desc}_1(n, d) \rightarrow \exists d'\ \mathtt{desc}_2(n', d') \wedge C(a, t, d, d')] \quad (3)$$

*Compiling relational-to-XML encodings.* Recall from Step 1 in section 1 that in order to uniformly express the schema mappings as XQueries, we encode the relational data as XML. Various schemes have been proposed, all having in common the fact that each relational tuple corresponds to an XML subtree whose nodes have fresh, invented identities. We have encountered a similar situation when compiling XQuery views, where the tuples were the variable bindings produced

[8] Many semistructured and XML query languages use functions like $F_{item}$ as explicit query primitives, under the name of Skolem functions. Our technique for compiling into dependencies fits seamlessly with an extension of XQuery with Skolem functions.

by the XBind queries and the XML trees were given by the element constructor. We therefore use similar DEDs to capture the encoding.

### 4.4 The Algorithm

If any variables of the XBind query $Xb$ are bound to element nodes, then $Xb$ cannot be reformulated against the storage schema: if the latter is relational, it contains no XML nodes, and if it is mixed, then the node identities in the storage and published data are disjoint. We hence need to find query "plans" which collect data from the storage but also *invent* and *copy* nodes, according to the semantics of the XQuery views that define the schema correspondence.

**Plans: reformulations using auxiliary schema.** We show in section 4.3 how to model this semantics using Skolem and copy functions. Suppose a plan retrieves the storage data tuples that satisfy condition $c(\mathbf{x})$ and returns $\mathbf{y}$ and an invented node $n = F(\mathbf{z})$ where $F$ is a Skolem function and $\mathbf{y}, \mathbf{z} \subseteq \mathbf{x}$. This plan can be described as the query $P(\mathbf{y}, n) \leftarrow c(\mathbf{x}), G(n, \mathbf{z})$, with $G$ the graph of $F$ $(G(n, \mathbf{z}) \Leftrightarrow n = F(\mathbf{z}))$. Denote with `Aux` the relational symbols modeling the graphs of Skolem and copy functions. Then any plan can be represented by a query against the extended storage schema $S \cup$ `Aux`.

*Algorithm* **for XBind reformulation.**
**Given:**
– an XBind query $Xb$ (obtained from a behaved XQuery)
– a schema correspondence described by a set of behaved XQuery views $\mathbf{V}$.
– the set $\mathbf{C_X}$ of `XIC`s over the various XML documents (public or storage)
– the set $\mathbf{C_R}$ of relational integrity constraints over the relational part of the storage schema $S$.
**Do:**
– Compile $Xb$ to the union of conjunctive queries $c(Xb)$
– Compile the schema correspondence to the set of DEDs $c(\mathbf{V})$. In the process, we introduce the set `Aux` of Skolem and copy function graphs (see section 4.3).
– Compile $\mathbf{C_X}$ to the set $c(\mathbf{C_X})$ of DEDs.
– Let $\mathbf{R}$ be the set of reformulations *against* $S \cup$ `Aux` obtained by applying the C&B algorithm to $c(Xb)$ under $\mathtt{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C_X}) \cup \mathbf{C_R}$.
**Return:**
– all queries in $\mathbf{R}$ that correspond to a viable reformulation plan. **End.**

**Bounded `XIC`s.** In [12], we introduce the class of *bounded* `XIC`s, such that we can guarantee the termination of the chase with $c(C_\mathbf{X})$. We also show there that containment of XBind queries is undecidable in the presence of `XIC`s that make even modest use of unboundedness. From proposition 2 it follows that no minimization algorithm is complete for unbounded `XIC`s. A bounded `XIC` allows existential quantification over element nodes, but only when their depth in the XML tree is bounded by the size of the `XIC`. The class is quite expressive, it contains XML Schema key constraints, many keyref constraints, and constraints implied by the content model definition of XML elements. In section 4.2, the `XIC` (2) is bounded, but $\forall x\ [//employee](x) \rightarrow \exists y\ [//employer](y)$ is not.

**Theorem 2 (Relative Completeness).** *If the constraints in* $\mathbf{C_X}$ *are bounded and* $\mathbf{C_R}$ *has stratified-witness, then* $R$ *is a minimal reformulation of* $Xb$ *if and only if* $c(R)$ *is a minimal reformulation of* $c(Xb)$ *under* $\mathtt{TIX} \cup c(\mathbf{V}) \cup c(\mathbf{C_X}) \cup \mathbf{C_R}$.

Theorems 2 and 1 immediately imply the following

**Corollary 1 (Overall Completeness).** *The algorithm finds all minimal reformulatons for behaved client XBind queries, under behaved XQuery views, bounded* XICs *and stratified-witness relational dependencies.*

*Remark.* It follows from proposition 2 that even modest use of non-behaved features such as wildcard child navigation results in an incomplete algorithm unless $NP = \Pi_2^p$: in [12] we show that containment for XBind queries with wildcard child is $\Pi_2^p$-hard even when the queries are disjunction-free and use no ancestor navigation. On the other hand, it turns out that the C&B gives us a reformulation in NP in the size of these queries.

## 5 Constraint Reformulation

We present a way to reuse any query reformulation algorithm for constraint reformulation, exploiting the following fundamental reduction between query containment and constraint satisfaction.

**Proposition 3.** *(a) For every* XIC $d$ *there are XBind queries* $Q_1^d, Q_2^d$ *such that for any instance* $I$, $I \models d \Leftrightarrow Q_1^d(I) \subseteq Q_2^d(I)$. *(b) For every XBind queries* $Q_1, Q_2$, *there is an* XIC $cont(Q_1, Q_2)$ *such that for every instance* $I$, $Q_1(I) \subseteq Q_2(I) \Leftrightarrow I \models cont(Q_1, Q_2)$.

*Proof:* (a) For $d$ of form $\forall \mathbf{x}\ [B(\mathbf{x}) \rightarrow \exists \mathbf{y}\ C(\mathbf{x}, \mathbf{y})]$, construct $Q_1^d(\mathbf{x}) \leftarrow B(\mathbf{x})$ and $Q_2^d(\mathbf{x}) \leftarrow B(\mathbf{x}) \wedge C(\mathbf{x}, \mathbf{y})$. (b) For $Q_1(\mathbf{x}) \leftarrow B_1(\mathbf{x}, \mathbf{y})$ and $Q_2(\mathbf{x}) \leftarrow B_2(\mathbf{x}, \mathbf{z})$ , $cont(Q_1, Q_2) = \forall \mathbf{x} \forall \mathbf{y}\ [B_1(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ B_2(\mathbf{x}, \mathbf{z})]$ .

**XIC reformulation algorithm:** (1) construct $Q_1^d, Q_2^d$, (2) reformulate each against $S \cup \mathtt{Aux}$, to $R_1$, resp. $R_2$, (3) construct $cont(R_1, R_2)$, and (4) return the restriction of $cont(R_1, R_2)$ to $S$. •

Since in general $d$ quantifies over XML nodes (recall XIC (2) in section 4.2), $Q_1^d, Q_2^d$ cannot be reformulated against the storage schema $S$ only, as it does not contain these nodes. However, $Q_1^d, Q_2^d$ are XBind queries, which we can reformulate against $S \cup \mathtt{Aux}$. By the following result, we can always turn $cont(R_1, R_2)$ (against $S \cup \mathtt{Aux}$) into a dependency formulated solely against $S$:

**Proposition 4.** *Let* $d_R$ *be obtained from* $cont(R_1, R_2)$ *by simply dropping all atoms involving any variable* $x$ *appearing as the result of a function from* Aux. *Then on all instances satisfying the schema correspondence,* $cont(R_1, R_2)$ *is satisfied if and only if* $d_R$ *is.*

Notice that by "plugging in" any sound query reformulation algorithm, we obtain a sound algorithm for constraint reformulation. Details are provided in [13,11].

## 6 Summary

We have presented an algorithm for finding the minimal reformulations of client XQueries in XML publishing scenarios, when the correspondence between public and storage schema is given by a combination of GAV and LAV XQuery views. The algorithm handles in the same unified way redundant storage (typical in XML applications), constraints in XML data (as specified by XML Schema) and constraints in the relational storage. The algorithm is complete and asymptotically optimal for an expressive class of client query and views (behaved XQueries) and integrity constraints (bounded `XIC`s and stratified-witness DEDs). The algorithm can be reused for reformulation of `XIC`s. Given its direction-independence, it applies also to reformulating integrity constraints on the storage to constraints on the public schema. This is useful for publishing integrity constraints to help clients understand the semantics of the published data.

**Practicality of the approach.** We have built a query reformulation system [11] based on the method presented here. Putting these ideas to work required a good deal of challenging engineering but, as we plan to report elsewhere, the performance of the resulting system proves that the method is definitely practical.

**Acknowledgements.** We are very grateful to Lucian Popa for his contributions. We also thank Dan Suciu, Mary Fernandez, Susan Davidson, Peter Buneman, Yi Chen and Yifeng Zheng for their useful suggestions.

## References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. A. Aho, Y. Sagiv, and J. Ullman. Efficient optimization of a class of relational expressions. In *TODS*, 4(4), 1979.
3. C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *JACM*, 31(4):718–741, 1984.
4. P. Buneman, S. Davidson, W. Fan, C, Hara, and W.-C. Tan. Keys for xml. In *WWW10*, May 2001.
5. A. Cali, G. De Giacomo, and M. Lenzerini. Models of information integration: Turning local-as-view into global-as-view. In *FMII*, 2001.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *PODS*, 1999.
7. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. View-based query processing for regular path queries with inverse. In *PODS*, 2000.
8. M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: Middleware For Publishing Object-Relational Data as XML Documents. In *VLDB*, Sep 2000.
9. A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *SIGMOD*, 1999.
10. A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*, 1999.

11. A. Deutsch. XML Query Reformulation Over Mixed and Redundant Storage. *PhD thesis*, University of Pennsylvania, 2002. Available from `http://db.cis.upenn.edu/cgi-bin/Person.perl?adeutsch`
12. A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath Fragments. In *KRDB*, 2001.
13. A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints (extended version). Available from `http://db.cis.upenn.edu/cgi-bin/Person.perl?adeutsch`
14. A. Deutsch and V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *DBPL*, 2001.
15. M. Fernandez, A. Morishima, and D. Suciu. Efficient Evaluation of XML Middleware Queries. In *SIGMOD*, 2001.
16. M. Fernandez, W. Tan, and D. Suciu. SilkRoute: Trading between Relations and XML. In *WWW9*, 2000.
17. M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, 1999.
18. J. Goldstein and P. A. Larson. Optimizing queries using materialized views. In *SIGMOD*, 2001.
19. J. Gryz. Query folding with inclusion dependencies. In *ICDE*, 1998.
20. A. Halevy. Logic-based techniques in data integration. In *Logic Based Artificial Intelligence*, 2000.
21. R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *VLDB*, 1990.
22. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.
23. A. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.
24. A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, 1996.
25. I. Manolescu, D. Florescu, and D. Kossman. Answering XML Queries on Heterogeneous Data Sources. In *VLDB*, 2001.
26. Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructured Data, In *SIGMOD*, 1999.
27. L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, Univ. of Pennsylvania, 2000.
28. R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. In *VLDB Journal*, 10(2-3), 2001.
29. P. Seshadri, H. Pirahesh, and T. Y. C. Leung. Complex query decorrelation. In *ICDE*, 1996.
30. J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *VLDB*, 2001.
31. O. Tsatalos, M. Solomon, and Y. Ioannidis. The gmap: A versatile tool for physical data independence. *VLDB*, 1994.
32. W3C. XML Schema Part 0: Primer. Working Draft 25 February 2000. Available from `http://www.w3.org/TR/xmlschema-0`.
33. W3C. XQuery: A query Language for XML. W3C Working Draft 15 February 2001. Available from `http://www.w3.org/TR/xquery`.

# New Rewritings and Optimizations for Regular Path Queries

Gösta Grahne and Alex Thomo

Concordia University,
Montreal, Canada
{grahne, thomo}@cs.concordia.ca

**Abstract.** All the languages for querying semistructured data and the web use as an integral part regular expressions. Based on practical observations, finding the paths that satisfy those regular expressions is very expensive. In this paper, we introduce the "maximal partial rewritings" (MPR's) for regular path queries using views. The MPR's are always exact and more useful for the optimization of the regular path queries than other rewritings from previously known methods. We develop an algorithm for computing MPR's and prove, through a complexity theoretic analysis, that our algorithm is essentially optimal. Also, we present query answering algorithms that utilize exact partial rewritings for regular path queries and conjunctive regular path queries respectively.

## 1 Introduction

Semistructured data are self-describing collections, whose structure can naturally model irregularities that cannot be captured by relational or object-oriented data models [1]. This kind of data is usually best formalized in terms of labeled graphs, where the graphs represent data found in many useful applications such as web information systems, XML data repositories, digital libraries, communication networks, and so on. Virtually, all the query languages for semi-structured data provide the possibility for the user to query the database through regular expressions. The design of query languages using regular path expressions, is based on the observation that many of the recursive queries that arise in practice amount to graph traversals. These queries are in essence graph patterns, and the answers to the query are subgraphs of the database that match the given pattern [17,7, 3,4].

For example, for answering a query containing in it the regular expression $(\_^* \cdot \mathit{article}) \cdot (\_^* \cdot \mathit{ref} \cdot \_^* \cdot (\mathit{abiteboul} + \mathit{vianu}))$, one should find all the paths having at some point an edge labeled *article*, followed by any number of other edges, then by an edge *ref*, and finally by an edge labeled with *abiteboul* or *vianu*.

Based on practical observations, the most expensive part of answering queries on semistructured data is to find those graph patterns described by regular expressions. This is because a regular expression can describe arbitrarily long paths in the database, which means in turn an arbitrary number of physical accesses. Hence, it is clear that having a good optimizer for answering regular

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 242–258, 2003.

path queries is very important. This optimizer can be used for the broader classes of query languages for semistructured data, as we illustrate in the paper.

In semistructured data as well as in other data models, such as relational and "object oriented," the importance of utilizing views is well recognized [16, 3,15,18]. Simply stated the problem is: Given a query $Q$ and a set of views $\{V_1, \ldots, V_n\}$, find a representation (rewriting) of $Q$ by means of the views and then answer the query on the basis of this representation.

The most notable methods for obtaining rewritings for regular path queries are by Calvanese *et al* in [3,5,6]. However, these methods rewrite –using views– only complete words of the query. But in practice, the cases in which we can infer from the views full words for the query, are very "ideal." The views can cover *partial* words that can be satisfactorily long for using them in optimization, but if they are not complete words, they are ignored by the above mentioned methods. It would however be desirable to have a partial rewriting in order to capture and utilize all the information provided by the views.

The problem of computing a partial rewriting is initially treated by Calvanese *et at* in [3]. There this problem is considered as an extension of the complete rewriting, enriching the set of the views with new elementary one-symbol views, chosen among the database relations (or symbols). The choice of the new elementary views is done in a brute force way, using a cost criterion depending on the application. However, there are cases when the algorithm of [3] for computing partial rewritings gives "too much" (in a sense to be made precise later). It essentially contains redundant un-rewritten (sub)words from the query.

In a previous paper [10], the present authors presented another algorithm for computing contained partial rewritings. Using that algorithm we avoid getting "too much," but unfortunately, the rewriting is not guaranteed to be exact, which diminishes its usability.

In this paper we will introduce the "maximal partial rewritings" (MPR's) for regular path queries using views. We will show that they don't give "too much," that they are always exact and more useful for the optimization of the regular path queries.

Then, we will present an algorithm that, when using exact partial rewritings, optimizes the evaluation of regular path queries to a database. We also explore the use of the partial rewritings for the optimization of the wider class of conjunctive regular path queries (CRPQ's). We introduce the "conjunctive exact partial rewritings" (CEPR's) and present an algorithm for utilizing them in CRPQ evaluation.

Finally, through a complexity theoretic analysis, we prove that our algorithm for computing the "maximal partial rewritings" is essentially optimal.

Due to space limitations, the proofs of some theorems and lemmas are omitted. These proofs appear in the full version of the paper [12].

## 2 Background

**Semistructured databases.** We consider a database to be an edge labeled graph. This graph model is typical in semistructured data, where the nodes of

the database graph represent the objects and the edges represent the attributes of the objects, or relationships between the objects.

Formally, we assume that we have a universe of objects $D$ and a finite alphabet $\Delta$, called the *database alphabet*. Objects of $D$ will be denoted $a, b, c, a', b', \ldots, a_1, b_2, \ldots$, etc. Elements of $\Delta$ will be denoted $R, S, T, R', S', \ldots, R_1, S_1, \ldots$, etc.

A *database* $DB$ over $(D, \Delta)$ is a pair $(N, E)$, where $N \subseteq D$ is a set of nodes and $E \subseteq N \times \Delta \times N$ is a set of directed edges labeled with symbols from $\Delta$.

**Queries and rewritings of queries using views.** A *regular path query* $Q$ is a finite or infinite regular language over the alphabet $\Delta$. Let $Q$ be a regular path query, and $DB = (N, E)$ a database. Then the *answer to* $Q$ on $DB$ is defined as:

$$ans(Q, DB) = \{(a, b) \in N^2 \ : \ a \xrightarrow{W} b \text{ for some } W \in Q\}.$$

Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of *view definitions* with each $V_i$ being a finite or infinite regular language over $\Delta$. Associated with each view definition $V_i$ there is a view name $v_i$. We call the set $\Omega = \{v_1, \ldots, v_n\}$ the *outer* or *view alphabet*. For each $v_i \in \Omega$, we set $def(v_i) = V_i$. The substitution *def* associates with each view name $v_i$, in the $\Omega$ alphabet, the language $V_i$. Also, we extend the substitution *def* to the $\Delta$ alphabet associating each symbol with itself. The substitution *def* is applied to words, languages, and regular expressions, over $\Omega \cup \Delta$ in the usual way (see e. g. [13]). Sometimes we need to refer to regular expressions representing the languages $Q$ and $V_i$. In order to simplify the notation, we will blur the distinction between the regular expressions and the languages that they represent.

The *maximally contained rewriting*, of a user query $Q$ using $\mathbf{V}$ ($MCR_{\mathbf{V}}(Q)$), is the language $Q'$ over $\Omega$ that includes *all* the words $v_{i_1} \ldots v_{i_k} \in \Omega^*$, such that

$$def(v_{i_1} \ldots v_{i_k}) \subseteq Q.^1$$

The $MCR_{\mathbf{V}}(Q)$ can be empty even if the desired answer is not. Suppose, for example, that the query $Q$ is $Q = R_1 \ldots R_{100}$ and we have available two views $V_1$ and $V_2$, where $V_1 = R_1 \ldots R_{49}$ and $V_2 = R_{51} \ldots R_{100}$. It is easy to see that $MCR_{\mathbf{V}}(Q)$ is empty. However, depending on the application, a "partial rewriting" such as $v_1 R_{50} v_2$ could be useful. In the next section we develop a formal algebraic framework for the partial rewritings. In sections 5 and 6 we demonstrate the usefulness of the partial rewritings in query optimization.

## 3 Partial Rewritings

Let $L$ be a language and $M$ an $\epsilon$-free language, both of them over some alphabet $\Delta$. Let $m$ be a symbol outside $\Delta$ and set $def(m) = M$. A *partial $M$-rewriting* of

[1] It is easy to see that this definition of $MCR_{\mathbf{V}}(Q)$ matches exactly the language of the output automaton from Theorem 1 in [3], which is the rewriting as defined there, with respect to the view alphabet.

$L$ is a language $L'$ over $\Delta \cup \{m\}$, such that $def(L') \subseteq L$. The partial rewriting is said to be *exact*, if $def(L') = L$. Since the (complete) rewritings, i.e. the ones that use only the $m$ symbol, are special cases of the partial rewritings, we will blur the distinction between them and call the partial rewritings just *rewritings*.

It is clear that there can be several $M$-rewritings of the same language $L$. In order to compare different rewritings, we introduce a partial order between the languages over $\Delta \cup \{m\}$. With this partial order we want to capture the intuition that the more subwords on the $\Delta$-alphabet that have been replaced by $m$ in a rewriting, the "bigger" (and "better") the rewriting.

Let $L_1$ and $L_2$ be languages over $\Delta \cup \{m\}$. We define $L_1$ to be $M$-smaller than $L_2$, denoted $L_1 \leq_M L_2$, if it is possible to substitute by $m$ some (not necessarily all) occurrences of words over $M$ occurring as subwords in $L_1$, and obtain $L_2$ as a result.

Obviously $\leq_M$ is transitive and reflexive. It is not antisymmetric, as for instance $\{mRR, mm, RRRR\} \leq_M \{mm, RRRR\}$, and $\{mRR, mm, RRRR\} \geq_M \{mm, RRRR\}$, when $M$ is for example $\{RR\}$. However, if we define $L_1 \equiv_M L_2$ iff $L_1 \leq_M L_2$ and $L_2 \leq_M L_1$, we will get a partial order on the equivalence classes.

Notably, we have that if a set $L'$ is $\leq_M$-maximal, then its equivalence class is singleton. To show this, we first present the following theorem.

**Theorem 1.** *Let $L'$ be a language on $\Delta \cup \{m\}$. Then $L'$ is $\leq_M$-maximal, if and only if, there does not exist a word $W \in L'$, such that $W = W_1W_2W_3$, and $W_2 \in M$.*

**Corollary 1.** *Let $L'$ be a language on $\Delta \cup \{m\}$. If $L'$ is $\leq_M$-maximal, then its $\equiv_M$-equivalence class is a singleton.*

Now, consider this partial order restricted to the set of all the $M$-rewritings of a language $L$. We will denote the restricted partial order with $\leq_M^L$. Obviously, we are interested in the $\leq_M^L$-maximal $M$-rewritings of $L$. With a similar reasoning as before, we can prove the next theorem and corollary. The main difference is that we cannot now replace just any subword which is a word in $M$. Naturally, a word $W = W_1W_2W_3$, where $W_2 \in M$ and $def(W_1mW_3) \subseteq L$, is not yet an "optimal" word, and we call $W_2$ a subword *eligible for replacement*.

**Theorem 2.** *Let $L'$ be a rewriting of $L$ on $\Delta \cup \{m\}$. Then $L'$ is $\leq_M^L$-maximal, if and only if, there does not exist a word $W \in L'$, such that $W = W_1W_2W_3$, where $W_2 \in M$, and $def(W_1mW_3) \subseteq L$.*

**Corollary 2.** *Let $L'$ be a rewriting on $\Delta \cup \{m\}$. If $L'$ is $\leq_M^L$-maximal, then its $\equiv_M^L$-equivalence class is a singleton.*

Also, we require exactness for a rewriting. Only then a rewriting becomes useful for query optimization [4,11].

A rewriting that is both $\leq_M^L$-maximal and exact is the union of all $\leq_M^L$-maximal $M$-rewritings of $L$. We call this rewriting the *maximal partial $M$-rewriting* of $L$, and denote it with $MPR_M(L)$.

From all the above, we can see that the $MPR_M(L)$ is the set of *all* the words on $\Delta \cup \{m\}$ with no subword for potential "contained replacement."

*Example 1.* Let $M = \{RSR, S\}$, and

$$L = \{RSRS, SRSR, RSRRSR, SS, SRSRS, SSS\}.$$

Then

$$MPR_M(L) = \{mm, SmS\}.$$

Here, $mm$ is obtained from $RSRS$, $SRSR$, $RSRRSR$, or $SS$. The word $SmS$ is obtained from $SRSRS$ or $SSS$. There are no more eligible subwords for replacement. For instance, replacing subwords in $\underline{S}m\underline{S}$ that belong to $M$, would violate the containment condition necessary for being a rewriting.

Formally, we have that:

**Theorem 3.** *The rewriting $MPR_M(L)$ is $\leq_M^L$-maximal and exact.*

We can give a natural generalization of the definition of the maximal partial M-rewriting for the case when we like to replace subwords not from one language only, but from a finite set of languages (such as a finite set of view definitions). For this purpose, suppose we are given the $\epsilon$-free view languages $\mathbf{V} = \{V_1, \ldots, V_n\}$ and a target query language $Q$, all of them over the alphabet $\Delta$. Also, consider the view alphabet $\Omega = \{v_1, \ldots, v_n\}$, for which as defined in Section 2, we have $def(v_i) = V_i$, for $i \in [1, n]$. A *partial* $\mathbf{V}$*-rewriting* of $Q$ is a language $Q'$ over $\Delta \cup \Omega$, such that $def(Q') \subseteq Q$. The partial rewriting is said to be *exact* if $def(Q') = Q$. Then, in order to compare the rewriting we define analogously the partial order $\leq_{\mathbf{V}}$ on $(\Delta \cup \Omega)^*$ as follows.

Let $Q_1$ and $Q_2$ be languages over $\Delta \cup \Omega$. We define $Q_1$ to be $\mathbf{V}$-smaller than $Q_2$, denoted $Q_1 \leq_{\mathbf{V}} Q_2$, if it is possible to substitute by $v_{i_1}, \ldots, v_{i_k}$ some (not necessarily all) occurrences of words over $V_{i_1}, \ldots, V_{i_k}$ respectively, occurring as subwords in $Q_1$, and obtain $Q_2$ as a result.

As before, we restrict this partial order to the set of all the $\mathbf{V}$-rewritings of a language $Q$. Similarly, we denote the restricted partial order with $\leq_{\mathbf{V}}^Q$. Finally, we define the *maximal partial* $\mathbf{V}$*-rewriting* of $Q$, denoted $MPR_{\mathbf{V}}(Q)$, to be the *union* of all $\leq_{\mathbf{V}}^Q$-maximal $\mathbf{V}$-rewritings $Q'$ of $Q$. Clearly, as in Theorem 3, we can show that the $MPR_{\mathbf{V}}(Q)$ is $\leq_{\mathbf{V}}^Q$-maximal and exact.

We will compare in the following the MPR's with the partial rewritings proposed in the literature. We begin with the partial rewriting of [3]. In that paper, candidate sub-alphabets $\Delta' \subseteq \Delta$ are selected using some cost criteria, and then the symbols of $\Delta'$ are considered as new elementary one-symbol views. Note that there are exponentially many candidate sub-alphabets. Each time, using the algorithm presented in [3], a rewriting is computed and after that its exactness is tested. However, the algorithm used, although very suitable for computing all the rewritings in the view alphabet $\Omega$, can compute "non-optimal" $\Delta \cup \Omega$ words when it is used for computing partial rewritings. As an example, consider the regular path query $Q = R_1R_2 + R_2^5$ and two views $V_1 = R_1$ and $V_2 = R_2^5$. Then,

the algorithm of [3] will give as partial rewriting $V_1R_2 + V_2 + R_2^5$, which has the redundant "non-optimal" word $R_2^5$. As a consequence, the partial rewriting of [3] is not always $\leq_{\mathbf{V}}^{Q}$-maximal. We call the partial rewriting of [3] the *maximally contained partial rewriting* or $MCPR_{\mathbf{V}}$ because it contains *all* the words $W$ over $\Delta' \cup \Omega$, such that $def(W) \subseteq Q$. Observe here that the maximality in this rewriting is with respect to the containment of $\Delta' \cup \Omega$ words and not really to their optimality regarding the non-view symbols.

Now, let's consider the partial rewriting presented in [11]. This rewriting is the union of all $\leq_{\mathbf{V}}$-maximal $\mathbf{V}$-rewritings of $Q$. By Theorem 1 we have that this is the set of all "mixed" words $W$ on the alphabet $\Omega \cup \Delta$ with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query $Q$. We call this set *exhaustive lower partial rewriting*, or $ELPR_{\mathbf{V}}(Q)$, because we replace subwords of $Q$ with view symbols thinking only about the optimality of words with regard to the non-view symbols, but not caring about the exactness of the rewriting. Clearly, since any $\leq_{\mathbf{V}}$-maximal rewriting is also $\leq_{\mathbf{V}}^{Q}$-maximal, the result is that we get always a $\leq_{\mathbf{V}}^{Q}$-maximal rewriting, but its exactness is not guaranteed.

Finally, $MPR_{\mathbf{V}}(Q)$ is a rewriting that satisfies both the optimality with regard to the non-view symbols and the exactness. Summarizing, and also including the maximally contained (complete) rewriting $MCR_{\mathbf{V}}$ in the comparison, we have the following table.

| | $\leq_{\mathbf{V}}^{Q}$-maximality | Exactness |
|---|---|---|
| $MCR$ [3] | YES | NO |
| $MCPR$ [3] | NO | YES |
| $ELPR$ [11] | YES | NO |
| $MPR$ | YES | YES |

## 4 Computing the Maximal Partial Rewriting

To this end, we will first give a characterization of the maximal partial $M$-rewriting of a language $L$. The construction in the proof of our characterization provides the basic algorithm for computing the rewriting $MPR_M(L)$ on a give regular language $L$.

The construction is based on finite transducers. A *finite transducer* $T = (S, I, O, \delta, s_0, F)$ consists of a finite set of states $S$, an input alphabet $I$, and an output alphabet $O$, a starting state $s_0$, a set of final states $F$, and a transition-output relation $\delta \subseteq S \times I^* \times S \times O^*$. Intuitively, for instance $(s_0, U, s_1, W) \in \delta$ means that if the transducer is in state $s_0$ and reads word $U$ it can go to state $s_1$ and emit the word $W$. For a given word $U \in I^*$, we say that a word $W \in O^*$ is an *output of* $T$ *for* $U$ if there exists a sequence $(s_0, U_1, s_1, W_1) \in \delta$, $(s_1, U_2, s_2, W_2) \in \delta$, $\ldots$, $(s_{n-1}, U_n, s_n, W_n) \in \delta$ of state transitions in $T$, such that $s_n \in F$, $U = U_1 \ldots U_n$, and $W = W_1 \ldots W_n$. We write $W \in T(U)$, where $T(U)$ denotes set of all outputs of $T$ for the input word $U$. For a language $L \subseteq I^*$ we define $T(L) = \bigcup_{U \in L} T(U)$. It is well known that $T(L)$ is regular whenever $L$ is.

We are now in a position to state our characterization theorem.

**Theorem 4.** *Let $L$ and $M$ be regular languages over an alphabet $\Delta$. There exists a finite transducer $T$ and a regular language $M'$, such that*

$$MPR_M(L) = (T(L^c))^c \cap M',$$

*where $(.)^c$ denotes set complement.*

*Proof.* Let $A = (S, \Delta, \delta, s_0, F)$ be a nondeterministic finite automaton that accepts the language $M$. Let us consider the finite transducer:

$$T = (S \cup \{s_0'\}, \Delta, \Delta \cup \{m\}, \delta', s_0', \{s_0'\}),$$

where $\delta'$, written as a relation, is

$$\begin{aligned}\delta' = &\{(s, R, s', \epsilon) : (s, R, s') \in \delta\} \cup \\ &\{(s_0', R, s_0', R) : R \in \Delta\} \cup \\ &\{(s_0', R, s, \epsilon) : (s_0, R, s) \in \delta\} \cup \\ &\{(s_0', R, s_0', m) : (s_0, R, s) \in \delta \text{ and } s \in F\} \cup \\ &\{(s, R, s_0', m) : (s, R, s') \in \delta \text{ and } s' \in F\}.\end{aligned}$$

Intuitively, transitions in the first set of $\delta'$ are the transitions of the "old" automaton, modified so as to produce $\epsilon$ as output. Transitions in the second set mean that "if we like, we can leave everything unchanged," i.e. each symbol gives itself as output. Transitions in the third set are for jumping non-deterministically from the new initial state $s_0'$ to the states of the old automaton $A$, that are reachable in one step from the old initial state $s_0$. These transitions give $\epsilon$ as output. Transitions in the fourth set are for handling special cases, when from the old initial state $s_0$, an old final state can be reached in one step. In these cases we can replace the one symbol words accepted by $A$ with the special symbol $m$. Finally, the transitions of the fifth set are the most significant. Their meaning is: in a state, where the old automaton has a transition by a symbol, say $R$, to an old final state, there will be, in the transducer, an additional transition $R/m$ to $s_0'$, which is also the (only) final state of $T$. Observe that, if the transducer $T$ decides to leave the state $s_0'$ while a suffix $U$ of the input string is unscanned, and enter the old automaton $A$, then it can return back only if there is a prefix $U'$ of $U$, such that $U' \in L(A)$. In such a case, the transducer $T$ replaces $U'$ with the special symbol $m$.

Given a word of $W \in \Delta^*$ as input, the finite transducer $T$ replaces arbitrarily many occurrences of words of $M$ in $W$ with the special symbol $m$. For an example, suppose $M$ is given by the automaton in Figure 1, top. The corresponding finite transducer is shown in the same figure, bottom. It consists of the automaton for $M$, whose transitions now produce as output $\epsilon$, plus the state $s_0'$, and the additional transitions, which are drawn with dashed arrows.

If $L'$ is a language on $\Delta$, it is straightforward to verify that

$$\begin{aligned}T(L') = L' \cup \{&U_1 m U_2 m \ldots m U_k : \\ &\text{for some } U \in L' \text{ and words } W_i \in M, \\ &U = U_1 W_1 U_2 W_2 \ldots W_{k-1} U_k\}.\end{aligned}$$

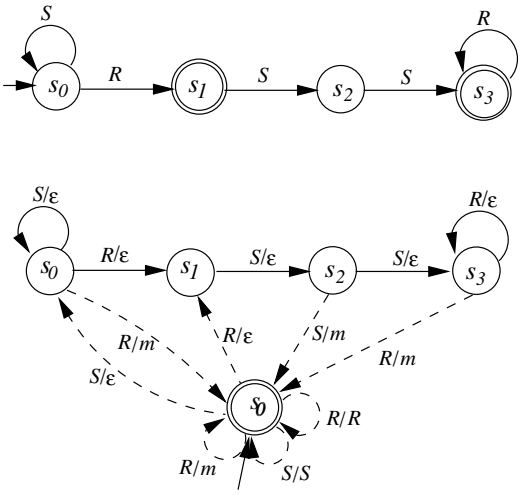



**Fig. 1.** An example of the construction for a replacement transducer

Recall that, we have $def(m) = M$, and $def(R) = R$ for each $R \in \Delta$. From the above, we can also easily characterize the set $T(L')$ from another point of view. Namely, $T(L')$ is the set of all words $W$ on $\Delta \cup \{m\}$, such that $def(W) \cap L' \neq \emptyset$.

Now, let's consider the transduction $T(L^c)$. As characterized above, $T(L^c)$ will be the set of all words $W$ on $\Delta \cup \{m\}$ such that $def(W) \cap L^c \neq \emptyset$. Hence, $T(L^c))^c$, being the complement of this set, will contain *all* the $\Delta \cup \{m\}$ words, such that *all* the $\Delta$-words in their substitution by *def* will be contained in $L$. This is the containment condition for a word on $\Delta \cup \{m\}$ to be in a rewriting. Clearly, $T(L^c))^c$ is a rewriting, and namely, it is the union of *all* the $M$-rewritings of $L$. Hence, $MPR_M(L) \subseteq T(L^c))^c$.

However, in order to compute $MPR_M(L)$, what we like is not a contained arbitrary, but a contained exhaustive replacement of words from $M$. To achieve this goal, we should filter out from the set $T(L^c)^c$, the words having eligible subwords for replacements (that do not violate the containment constraint). For this, consider the set $(\Delta \cup \{m\})^* M (\Delta \cup \{m\})^*$. This is the set of all the words on $\Delta \cup \{m\}$ that have at least one *potentially* eligible subword for replacement. But, as said before, this replacement should be "contained" in the language $L$. Formally speaking, we are interested in solving the following language equation. Find the biggest languages $X, Y \subseteq (\Delta \cup \{m\})^*$ such that

$$XMY \subseteq (T(L^c))^c.$$

Then, if we are able to find such $X$ and $Y$, we have that $XMY$ is the set of *all* the words on $\Delta \cup \{m\}$, that still have eligible subwords for replacement. Clearly, we filter out such words and have that

$$MPR_M(L) = (T(L^c))^c \cap (XMY)^c.$$

So, let's examine how to solve the above language equation. For this, consider another special symbol $m'$, such that $m' \notin \Delta \cup \{m\}$, and the substitution $def' : \Delta \cup \{m\} \cup \{m'\} \longrightarrow \Delta \cup \{m\}$, such that

$$def'(m') = M, \;\; def'(m) = m, \text{ and } \; def'(R) = R \text{ for } R \in \Delta.$$

Now, we build a transducer $T'$ in the same way as we did for the transducer $T$, but for the extended alphabet $\Delta \cup \{m\}$ and considering $m'$ as the special symbol. Reasoning similarly as before, if we transduce the complement of $(T(L^c))^c$ i.e. $T(L^c)$, we have that $(T'(T(L^c)))^c$ is the set of *all* words $W \in \Delta \cup \{m\} \cup \{m'\}$, such that $def'(W) \subseteq (T(L^c))^c$. However, in order to solve the above language equation, we will be interested in the subset of words in $(T'(T(L^c)))^c$ that contain exactly *one* special symbol $m'$. To get this subset, we filter out the un-wanted words by intersecting as in the following

$$(T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*.$$

The above is the set of *all* words $W = W_1 m' W_2$, where $W_1, W_2 \in (\Delta \cup \{m\})^*$, and $def'(W) \subseteq (T(L^c))^c$. Consider now the transducers $T_l$ and $T_r$ that erase from a word $W \in (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*$ the subword in $(\Delta \cup \{m\})^* m'$ and $m'(\Delta \cup \{m\})^*$ respectively. Finally, we set

$$X = T_r((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*),$$

and

$$Y = T_l((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*).$$

It is easy to see now that $X$ and $Y$ are regular languages and the maximal solution to the above equation. □

As a generalization of Theorem 4, we can give the following result about the maximal partial $\mathbf{V}$-rewriting, of a query $Q$ with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

**Theorem 5.** *Given a regular path query $Q$, the $MPR_{\mathbf{V}}(Q)$ can be effectively computed.*

## 5 Optimizing Regular Path Queries Using Partial Rewritings

In this section we show how to utilize partial rewritings in query optimization in a scenario where we have available a set of precomputed views, as well as the database itself. The views could be materialized views in a warehouse, or locally cached results from previous queries in a client/server environment. In this scenario, the views are assumed to be exact, and we are interested in answering the query by consulting the views as far as possible, and by accessing the database only when necessary.

Formally, let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions, and let $\Omega = \{v_1, \ldots, v_n\}$ be the view alphabet as before. We define the *view-graph* $\mathcal{V}$ to be a database over $(D, \Omega)$ induced by the set

$$\bigcup_{i \in \{1,\ldots,n\}} \{(a, v_i, b) : (a, b) \in ans(V_i, DB)\}.$$

of $\Omega$-labeled edges.

It has been shown in [4] that, given a query $Q$, if the $MCR_{\mathbf{V}}(Q)$ is exact, then $ans(Q, DB) = ans(MCR_{\mathbf{V}}(Q), \mathcal{V})$. However, the cases when we are able to obtain an exact complete rewriting of the query using the views could be rare in practice. In general, we have in the views only part of the information needed to answer the query. In the following, we will use exact partial rewritings not to *completely* avoid accessing the database, but to *minimize* such access as much as possible. We can use an exact partial rewriting $Q'$ to evaluate the query on the view-graph, accessing the database in a "lazy" fashion only when necessary.

The intuition behind our algorithm is that we use the view-graph as it was the database, and we extend it "on demand" as the navigation, guided by an automaton for $Q'$, requires. In fact, if we need to access a node in the database, we add to the view graph *all* (not only what we need) the outgoing edges and the neighboring nodes. This was motivated by considering the database as a set of HTML (or XML) pages related to each other through links. Accessing a page in the web could be time consuming, but subsequently, parsing the page text in the main memory for finding the links and the addresses, where these links point, is efficient. Since our queries could be recursive, a page (node) could be visited many times, and if it has already been fetched before from the database, along with all of its links, then there is no need to consult the database again. So, during the execution of the algorithm, for each page $x$ that we fetch from the database, we create a flag $Expanded_x$ and set it to true, meaning that now we have full local information for this page.

**Algorithm 1**
Input: An exact partial rewriting $Q'$ for a query $Q$ and a database $DB$.
Output: The answer to the query Q on database $DB$.
Method: First construct an automaton $A_{Q'}$ for $Q'$. Let $s_0$ be the initial state in $A_{Q'}$. Then, for each node $a \in N$, we compute a set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. For each transition $s_0 \xrightarrow{R} s$ in $A_{Q'}$, access $DB$ and add to $\mathcal{V}$ the subgraph of $DB$ induced by *all* the edges originating from the nodes $b$, which have some outgoing edges $R$. For each such node $b$, create a flag $Expanded_b$, and set it to **true**.
3. Repeat 4 until $Reach_a$ no longer changes.
4. Choose a pair $(b, s) \in Reach_a$.
   a) If there is a transition $s \xrightarrow{v_i} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{v_i} b'$ in $\mathcal{V}$, then add the pair $(b', s')$ to $Reach_a$.

b) Similarly, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, then add the pair $(b', s')$ to $Reach_a$.

c) Otherwise, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, but there is not an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, and there is not a flag $Expanded_b = \mathbf{true}$, then access the database and add to $\mathcal{V}$ the subgraph of $DB$ induced by *all* the edges originating from $b$. Create a flag $Expanded_b$, and set it to **true**.

Set $eval(Q', \mathcal{V}, DB) = \{(a, b) : (b, s) \in Reach_a$, and $s$ is a final state in $A_{Q'}\}$. □

It is easy to see that the following theorem is true.

**Theorem 6.** *Given a query $Q$ and a set $\mathcal{V}$ of exact cached views, using an exact partial rewriting $Q'$, we have that*

$$eval(Q', \mathcal{V}, DB) = ans(Q, DB).$$

## 6 Optimizing Conjunctive Regular Path Queries

A *conjunctive regular path query* (CRPQ) $Q$ is an expression of the form

$$Q(x_1, \ldots, x_l) : -y_1 E_1 z_1, \ldots, y_k E_k z_k,$$

where $x_1, \ldots, x_l$, $y_1, \ldots, y_k$, $z_1, \ldots, z_k$ are (not necessarily all distinct) variables over the universe of objects $D$, such that all the distinguished (head) variables $x_1, \ldots, x_l$ occur in the body, i. e. each $x_i$ is some $y$ or $z$, and $E_1, \ldots, E_k$ are regular languages (or regular path queries, RPQ) over the database alphabet $\Delta$. We call the conjunctions $yEz$ of a CRPQ, *regular path atoms*, or simply *atoms*.

The answer set $ans(Q, DB)$ to a CRPQ Q over a database $DB = (N, E)$ is the set of tuples $(a_1, \ldots, a_l)$ of nodes of DB, such that there is a total mapping $\tau$ from $y_1, \ldots, y_k, z_1, \ldots, z_k$ to $N$ with $\tau(x_i) = a_i$ for every distinguished variable $x_i$ of Q, and $(\tau(y), \tau(z)) \in ans(E, DB)$ for every atom $yEz$ in $Q$.

We say for an atom $yEz$, when $\tau(y) = a$ and $\tau(z) = b$, that $y$ and $z$ have been *bound* to the objects $a$ and $b$ respectively.

Suppose now, that we have available a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of conjunctive regular path views, which are defined as

$$V_i(x_{i1}, \ldots, x_{il_i}) : -y_{i1} E_{i1} z_{i1}, \ldots, y_{ik_i} E_{ik_i} z_{ik_i},$$

for $i \in [1, n]$. Let $\mathbf{E}$ be the set of all $E_{ij}$ above, for $i \in [1, n]$, and $\Omega$ be a set of $e_{ij} \notin \Delta$ corresponding symbols. Then, we define the *conjunctive exact partial* $\mathbf{V}$*-rewriting*, or $CEPR_{\mathbf{V}}(Q)$, of a CRPQ $Q$ with respect to a set $\mathbf{V}$ of conjunctive regular path view definitions, as

$$CEPR_{\mathbf{V}}(x_1, \ldots, x_l) : -y_1 E'_1 z_1, \ldots, y_k E'_k z_k,$$

where $E'_i$, for $i \in [1, k]$, is an *exact* partial rewriting of the regular language $E_i$ with respect to $\mathbf{E}$, with $\Omega$ as the corresponding set of special symbols.

Suppose that the sub-mechanism for answering the regular path atoms of the CRPQ's remembers the regular expressions and caches the corresponding answer sets of the regular path atoms in the recently processed CRPQ's. Observe that the cached answer sets of the regular path atoms $E_{ij}$ do not necessarily contain the full set $ans(E_{ij}, DB)$, but only those pairs that were called for, by the sideways information passing mechanism used. To formalize the "fullness" of a cached answer set, we introduce the notions of the "global completeness" and "local completeness," for the subsets of $ans(E, DB)$, for some $E$.

Let $P$ be a subset of $ans(E, DB)$, for some $E$ and $DB$. Then $P$ is said to be *globally complete* if $P = ans(E, DB)$. On the other hand, $P$ is *locally complete with respect to a node* $a$, if $P \supseteq \sigma_{\$1=a}(ans(E, DB))$. If $P$ is locally complete wrt all nodes in $\pi_{\$1}(P)$, we say that $P$ is *locally complete*[2].

We observe that, if the query processor can traverse the database graph only in the forward direction (as is the case in the web, see [2]), then the cached answer set of any atom $yE_{ij}z$, for which $z$ is not already bound to some objects, is locally complete. For simplicity, we call the atoms *locally complete*, when we have computed for them locally complete answer sets. As we will see, the locally complete atoms can be very useful in the query optimization.

Now, let's consider the atoms, which have the $z$ variable already bound to some objects. We observe that, for such atoms, we could have computed locally incomplete answer sets. In order to see this, suppose for example, that the variable $z$ has been bound to the objects $b$ and $c$. Then, if the variable $y$ bounds to an object, say $a$, the RPQ answering sub-mechanism using "cut"-like constructs, can stop the evaluation after computing in the answer, starting to navigate from $a$, the objects $b$ and $c$. But, the object $a$ could be connected with paths spelling words in the same regular language, to other database objects besides $b$ and $c$.

However, if the variable $z$ has been also bound to another object, say $d$, which cannot be reached from $a$, by following a path spelling a word in $E_{ij}$, then we inevitably have to compute a locally complete answer set for the (sub)atom $aE_{ij}z$, in order to reject the pair $(a, d)$. I such a case, the "cut"-like constructs do not have have a chance to execute. So, $aE_{ij}z$ is locally complete, and we cache its answer set for future optimizations.

Now, let's see how we can optimize the answering of the conjunctive regular path queries using conjunctive exact partial rewritings.

Consider the regular path atom $yE_{ij}z$ in some recently processed view $V_i$. As explained before, if the variable $z$ is not bound to some database object, then $yE_{ij}z$ is locally complete. Otherwise, we consider all the (sub)atoms $aE_{ij}z$, which are locally complete. Let's denote with $cache_{ij}$, the cached answer set of the locally complete atom $yE_{ij}z$, or if otherwise, the union of the cached answer sets of its locally complete (sub)atoms. Then, we define the *atom-view-graph* $\mathcal{V}$ to be a database over $(D, \Omega)$ induced by the set

[2] We consider for the simplicity of notations a set of object pairs as a binary relational table.

$$\bigcup_{i\in\{1,\ldots,n\}} \{(a, e_{ij}, b) : (a, b) \in cache_{ij}\}.$$

of $\Omega$-labeled edges.[3]

The evaluation will alternate between the database-graph and atom-view-graph. Suppose that we want to answer the atom $yEz$ and $y$ has already been bound to a node, say $a$. We now need to compute the nodes reachable from $a$, by a path spelling a word in $E$. Recall that for $E$ we have computed a corresponding exact rewriting $E'$. We start by answering $aE'z$ on $DB$. Intuitively, when during the navigation we are in a node $b$, and the regular expression for $E'$ requires an $\Omega$-symbol, say $e_{ij}$, to be matched, then we look at the atom-view-graph $\mathcal{V}$. If there is a node $b$ in $\mathcal{V}$, and in that node we find outgoing $e_{ij}$ edges, then we advance the navigation in the atom-view-graph. Since the cached answer for $E_{ij}$ is locally complete, we are sure that we get all the answers had we answered $E_{ij}$ directly in the database starting from $b$.

On the other hand, if we do not find an object $b$ in $\mathcal{V}$, or even if we find $b$ in $\mathcal{V}$, but there are no outgoing edges labeled with $e_{ij}$, then we evaluate $E_{ij}$, starting from $b$ in the database graph, and enrich the atom-view-graph accordingly. Clearly, we do not add by this operation any overhead for answering parts of some new $\Delta^*$ word, that is not in $E$, because the rewriting is exact. Formally, we have the following algorithm for answering $aEz$ on the basis of $E'$.

**Algorithm 2**
Input: An exact rewriting $E'$ for the (sub)atom $aEz$, and a database $DB$.
Output: The answer to the (sub)atom $aEz$ on database $DB$.
Method: First construct an automaton $A_{E'}$ for $E'$. Let $s_0$ be the initial state in $A_{E'}$. We compute the set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. Repeat 3 until $Reach_a$ no longer changes.
3. Choose a pair $(b, s) \in Reach_a$.
   a) If there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, then add the pair $(b', s')$ to $Reach_a$.
   b) Otherwise, if there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, but there is not an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, answer $bE_{ij}z$ on $DB$ and enrich $\mathcal{V}$ accordingly.
   c) If there is a transition $s \xrightarrow{R} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{R} b'$ in the database $DB$, then add the pair $(b', s')$ to $Reach_a$.

Finally, set $eval(E', a, DB) = \{(a, b) : (b, s) \in Reach_a$, and $s$ is a final state in $A\}$. □

It is easy to see that the following theorem is true.

[3] From another point of view, we can also see that a regular path atom (or (sub)atom), along with its locally complete answer set, is nothing else but a local node constraint as defined in [2].

**Theorem 7.** *Given a CRPQ $Q$ and a set $\mathbf{V}$ of views as above, using the rewriting $CEPR_{\mathbf{V}}(Q)$, we have that for some regular atom $E$ in $Q$, the answer to $aEz$ equals $eval(E', a, DB)$.*

Now, for the case of regular path atoms $aEz$, where the variable $z$ has already been bound to some objects, we slightly modify the above algorithm to evaluate the answer to such atoms as well. For this, let $B$ be the set of objects where $z$ has been bound. Then, in the above algorithm we *incrementally* compute in each step the set $eval(E', a, DB)$ and change the loop (2) to

Repeat 3 until $Reach_a$ no longer changes or $\pi_{\$2}(eval(E', a, DB))$ equals $B$.

We note that, the second condition of the loop termination is there for restricting the search space, in case we find (or verify) that we can reach from $a$ *all* the objects in $B$ by following paths, which spell words in $E$. Clearly, in this case, the answer set of the atom $aEz$ equals $a \times B$. On the other hand, we also stop if $Reach_a$ reaches a fixed point, and in this case the answer to the atom $aEz$ equals $a \times (\pi_{\$2}(eval(E', a, DB)) \cap B)$. In such a case, although the variable $z$ had already been bound, the set $eval(E', a, DB))$ is fully computed and so, it is locally complete.

Finally, if we set $B = \emptyset$, when the variable $z$ has not been already bound to some objects, then we can have a single *RPQ-answer-and-cache* algorithm. This algorithm would compute the answer set to an atom $yEz$, by iterating over all the objects $a$, where the variable $y$ could be bound, and compute $aEz$ by using an exact partial rewriting $aE'z$, as described above. At the end, we check whether the set $Reach_a$ has reached a fixed point. If yes, then we cache the locally complete $eval(E', a, DB))$, for future RPQ optimizations.

## 7 Complexity Analysis

**Theorem 8.** *Given a language $L$ and an $\epsilon$-free language $M$ both of them over an alphabet $\Delta$, the problem of generating the $MPR_M(L)$ is in 3EXPTIME.*

From the above theorem, we can easily derive the following corollary, regarding the upper complexity bound for the generation of the $MPR_{\mathbf{V}}(Q)$ of a query $Q$, with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

**Corollary 3.** *Given a language $Q$ and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of generating the $MPR_{\mathbf{V}}(Q)$ is in 3EXPTIME.*

We emphasize here that the above complexity analysis is a worst case analysis. In fact, the above complexity comes from possible DFA's state "blow up." However, despite these worst case possibilities, experimental results in [8], for converting graphs into DFA's, are encouraging, indicating that for the majority of the cases, the running time is very reasonable and the resulting DFA's are significantly smaller than their sources. Also, observe that if the probability of getting an exponential size DFA for an NFA is $p << 1$, then the probability of getting a triple exponential "blow up" in our case would roughly be $p^3$, which is very small.

Now, in order to prove that the above established upper bound is essentially optimal we will need the following constructions and theorems. Consider the set

$$(T(Q^c))^c \cap \left((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*\right)^c,$$

where the transducer $T$ is defined as in Theorem 5. This is in essence the *exhaustive lower partial rewriting*, or $ELPR_{\mathbf{V}}(Q)$ of [11], which is also discussed in Section 3. It is the set of all "mixed" words $W$ on the alphabet $\Omega \cup \Delta$, with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query $Q$. Obviously, $def(ELPR_{\mathbf{V}}(Q)) \subseteq Q$ and we are interested in the complexity of testing its exactness with respect to the query $Q$ [4]. We prove the following theorem regarding the upper bound of the above exactness problem.

**Theorem 9.** *Given a query $Q$ and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of testing $def(ELPR_{\mathbf{V}}(Q)) = Q$ is in 2EXPSPACE.*

*Proof.* By Theorem 8 the automaton $(T(Q^c))^c$ can exponentially "blow up" and namely be of doubly exponential size. The automaton for

$$\left((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*\right)^c$$

can also exponentially "blow up" but its size can be up to single exponential. So, in total the automaton for $ELPR_{\mathbf{V}}(Q)$ can be of up to doubly exponential size. Now, let's consider $def(ELPR_{\mathbf{V}}(Q))$. For the substitution *def* there exists a polynomial size transducer $T_{def}$ such that $def(ELPR_{\mathbf{V}}(Q)) = T_{def}(ELPR_{\mathbf{V}}(Q))$ [19,10]. Thus, the size of the automaton $T_{def}(ELPR_{\mathbf{V}}(Q))$ will be polynomial on the size of the automaton for $ELPR_{\mathbf{V}}(Q)$ i.e. it can be doubly exponential on the size of the automaton for $Q$. Now, to test the exactness $def(ELPR_{\mathbf{V}}(Q)) = Q$ is in PSPACE with regard to the size of the automaton for $ELPR_{\mathbf{V}}(Q)$. So, in total we have that testing the exactness of $ELPR_{\mathbf{V}}(Q)$ is in 2EXPSPACE. □

For the lower bound of the exactness $def(ELPR_{\mathbf{V}}(Q)) = Q$ we will use the maximally contained rewriting, $MCR_{\mathbf{V}}(Q)$ of [3]. As mentioned in Section 2, the $MCR_{\mathbf{V}}(Q)$ is the set of *all* words $W$ in $\Omega^*$, such that $def(W) \in Q$. In [3], the exactness problem for the $MCR_{\mathbf{V}}(Q)$ is proven to be 2EXPSPACE complete. We will give in the following a reduction of the exactness problem for $MCR_{\mathbf{V}}(Q)$ to the exactness problem for $ELPR_{\mathbf{V}}(Q)$.

Consider a query $Q$ and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions. Now, consider the set $\mathbf{V}' = \{V_1', \ldots, V_n'\}$ of new view definitions, where $V_i' = \$V_i\$$ for $i = [1, n]$ and $\$ \notin \Delta \cup \Omega$. Regarding the query $Q$, we will transform it into a new query $Q'$ through a transduction. For this we construct the following transducer.

Let $A_i = (S_i, \Delta, \delta_i, s_{0i}, F_i)$, for $i \in [1, n]$ be $n$ nondeterministic finite automata that accept the corresponding $V_i$ languages. Let us consider the finite transducer: $T_{\$\$} = (S_1 \cup \ldots \cup S_n \cup \{s_0'\}, \Delta, \Delta \cup \{\$\}, \delta', s_0', \{s_0'\})$, where

$$\delta' = \{(s, R, s', R) : (s, R, s') \in \delta_i,\ i \in [1, n]\} \cup$$

[4] The complexity bounds for testing the exactness $ELPR_{\mathbf{V}}(Q)$ are not discussed in [11].

$$\{(s'_0, \epsilon, s_{0i}, \$) : i \in [1, n]\} \cup$$
$$\{(s, \epsilon, s'_0, \$) : s \in F_i,\ i \in [1, n]\}.$$

The transducer $T_{\$\$}$ performs the following task: given a language $L$ as input, it produces as output all the words of $L$ having inserted in them the symbol \$ to mark the beginning and the end of a subword that belongs to $V_i$ for some $i \in [1, n]$. Moreover, only the words of $L$, which we can divide into a sequence of subwords, such that each belongs to some view language, are transduced. All the other $L$ words are filtered out. Formally,

$$T_{\$\$}(L) = \{\$U_1\$\$U_2\$ \dots \$U_k\$ : \text{for some } U \text{ in } L, U = U_1U_2 \dots U_k \text{ and } \forall i \in [1, k]\ \exists j \in [1, n] \text{ such that } U_i \in V_j\}.$$

**Theorem 10.** *Consider a query $Q$ and a set $\mathbf{V} = \{V_1, \dots, V_n\}$ of view definitions. Construct $Q$ and $\mathbf{V}' = \{V'_1, \dots, V'_n\}$ as above. Let $\Omega' = \{v'_1, \dots, v'_n\}$ and $def'$ be the usual view substitution on $\Omega'$, i.e. $def'(v'_i) = V'_i$, for $i \in [1, n]$. Also, let $def_{\$\to\epsilon}$ be the substitution that erases the symbol \$ from some language on $\Delta \cup \{\$\}$. Then, the following is true: the $MCR_{\mathbf{V}}(Q)$ is exact if and only if*

1. *$def_{\$\to\epsilon}(Q') = Q$, and*
2. *$ELPR_{\mathbf{V}'}(Q')$, with respect to $\mathbf{V}' = \{V'_1, \dots, V'_n\}$, is exact.*

Based on the above theorem we give the following theorem regarding the optimality of the construction given in Theorem 9 for testing the exactness of the ELPR of a query $Q$.

**Theorem 11.** *Given a query $Q$ and a set $\mathbf{V} = \{V_1, \dots, V_n\}$ of view definitions, the problem of testing $def(ELPR_{\mathbf{V}}(Q)) = Q$ is 2EXPSPACE complete.*

Finally, the following theorem shows the optimality of our algorithm for computing the MPR of a query using a set of view definitions.

**Theorem 12.** *The algorithm presented in Theorem 5 for computing the rewriting $MPR_{\mathbf{V}}(Q)$ of a query $Q$ given a set $\mathbf{V} = \{V_1, \dots, V_n\}$ of view definitions, is essentially optimal.*

*Proof.* Consider the $ELPR_{\mathbf{V}}(Q)$. Recall that this is the set of all "mixed" words $W$ on the alphabet $\Omega \cup \Delta$, with no subword in $V_1 \cup \dots \cup V_n$, such that their substitution by *def* is contained in the query $Q$. So, such words qualify for inclusion in $MPR_{\mathbf{V}}(Q)$. Hence, we have that $ELPR_{\mathbf{V}}(Q) \subseteq \text{CEPR}(Q)$. On the other hand, observe that if $ELPR_{\mathbf{V}}(Q)$ is *exact* then we must have $ELPR_{\mathbf{V}}(Q) = MPR_{\mathbf{V}}(Q)$. Now, we can test the exactness of $ELPR_{\mathbf{V}}(Q)$ by testing the condition $MPR_{\mathbf{V}}(Q) \subseteq ELPR_{\mathbf{V}}(Q)$, which is equivalent with the non emptiness of $MPR_{\mathbf{V}}(Q) \cap (ELPR_{\mathbf{V}}(Q))^c$. Since, by Theorem 9, we get a DFA for $ELPR_{\mathbf{V}}(Q)$, to complement does not add anything to the complexity of testing the emptiness of the above intersection. Now, if we were able to generate

the $MPR_{\mathbf{V}}(Q)$ in, say, 2EXPTIME, and since we are able to generate the $ELPR_{\mathbf{V}}(Q)$ in 2EXPTIME, then we could test the exactness of $ELPR_{\mathbf{V}}(Q)$ in 2EXPTIME, which is impossible by Theorem 11, unless 2EXPTIME = 2EXPSPACE. □

## References


1. S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
2. S. Abiteboul, V. Vianu. Regular Path Queries with Constraints. *Journal of Computing and System Sciences* 58(3) 1999, pp. 428–452
3. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Proc. of PODS* 1999, pp. 194–204.
4. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of ICDE* 2000, pp. 389–398.
5. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing for Regular Path Queries with Inverse. *Proc. of PODS* 2000, pp. 58–66.
6. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. What is View-Based Query Rewriting? *Proc. of KRDB* 2000, pp. 17–27.
7. D. Florescu, A. Y. Levy, D. Suciu Query Containment for Conjunctive Queries with Regular Expressions *Proc. of PODS* 1998, pp. 139–148.
8. R. Goldman and J. Widom. DataGuides: Enabling query Formulation and Optimization in Semistructured Databases. *Proc. of VLDB* 1997 pp. 436–445.
9. G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. *Proc. of ICDT* 1999 pp. 332–347.
10. G. Grahne and A. Thomo. An Optimization Technique for Answering Regular Path Queries. *Proc. of WebDB* 2000.
11. G. Grahne and A. Thomo. Algebraic Rewritings for Optimizing Regular Path Queries. *Proc. of ICDT* 2001 pp. 301–315.
12. G. Grahne and A. Thomo. New Rewritings and Optimizations for Regular Path Queries. `www.cs.concordia.ca/~faculty/grahne/papers/fullicdt03.ps`
13. J. E. Hopcroft and J. D. Ullman *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley 1979.
14. L. Kari. *On Insertion and Deletion in Formal Languages.* Ph.D. Thesis, 1991, Department of Mathematics, University of Turku, Finland.
15. A. Y. Levy. *Answering queries using views: a survey.* Technical Report, Computer Science Dept., Washington Univ., 2000.
16. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *Proc. of PODS* 1995, pp. 95–104.
17. A. O. Mendelzon and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp. 24:6,* (December 1995).
18. Y. Papakonstantinou, V. Vassalos. Query Rewriting for Semistructured Data. Proc. of SIGMOD 1999, pp. 455–466
19. S. Yu. Regular Languages. In: *Handbook of Formal Languages.* G. Rozenberg and A. Salomaa (Eds.). Springer Verlag 1997, pp. 41–110

# Database Interrogation Using Conjunctive Queries*

Michał Bielecki[1] and Jan Van den Bussche[2]

[1] Warsaw University, Poland, mab@mimuw.edu.pl
[2] University of Limburg, Belgium, jan.vandenbussche@luc.ac.be

**Abstract.** We consider a scenario where a client communicates with a database server by posing boolean conjunctive queries, or more generally, counts of conjunctive queries. We investigate to what extent features such as quantification, negation, or non-equalities are important in such a setting. We also investigate the difference between a setting where the client can pose an adaptive sequence of queries, and a setting where the client must pose a fixed combination of queries.

## 1 Introduction

Consider a scenario where a client program communicates with a database server through some query language $L$ that is rather limited. Due to the weakness of $L$, the query $Q$ the client really wants to ask is then often not expressible by a single query in $L$. Still the client may be able to derive the answer to $Q$ by performing some computation during which a *sequence* of queries, rather than a single one, is posed. We could call this "interrogation" rather than querying. Given the ubiquity of client-server access to databases, studying database interrogation appears to be well-motivated in general. A concrete example of interrogation occurs in database security [5,3].

Another well-known example of interrogation occurs in information integration using views [12]. There, $L$ is given by a finite number of conjunctive queries called views (possibly parameterized with selection constants, so that $L$ is actually infinite), and $Q$ is another conjunctive query, not in $L$. To answer $Q$, one tries to derive a conjunctive query $Q'$ over the views that is equivalent to $Q$. Executing this $Q'$ is then the computation performed by the client, and indeed involves posing to the database a sequence of queries, namely, the views involved in $Q'$.

In the above example, the sequence of queries the client poses is *database-independent*; $Q'$ depends only on $Q$ and $L$ but not on the database contents. One can also think of scenarios where the sequence is *adaptive* in that the choice of the next query to be posed depends on the actual answer received to the previous query (and thus depends on the database contents).

In the present paper, we study the more general setting where $L$ is the language of *all* conjunctive queries, or a variation thereof. The big difference with

* Research supported by Polish KBN grant 7T11C 007 21.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 259–269, 2003.

information integration is that the client can now pose any conjunctive query he likes, rather than only those provided as views. However, we have to be careful or this setting becomes trivial: the identity query, which simply downloads the entire database, is a conjunctive query, so a client needs to pose only that single query and then has enough information to compute the answer to any query he wants. Also from a practical server workload or network bandwidth point of view, downloads may be undesirable. Hence, we focus our attention to a setting where the server returns only boolean answers (i.e., the emptiness of the answer), or more generally, counts (i.e., the cardinality of the answer). For example, many search engines on the Internet directly return the cardinality of the search result.

In this setting, we study the effects of features such as quantification, negation, or non-equalities, and we compare database-independent client strategies to adaptive ones. Our contributions can be summarized as follows.

1. For database-independent strategies, non-equality tests matter. More concretely, against a server that answers counts of conjunctive queries, there is a simple conjunctive query extended with a non-equality test whose cardinality cannot be derived in a database-independent way.
2. The same holds for quantification: against a server that answers counts of quantifier-free conjunctive queries, there is a simple conjunctive query whose cardinality cannot be derived in a database-independent way.
3. Complementing the previous negative result with a positive result, we show that against the same server answering counts of quantifier-free conjunctive queries, we can derive the count of any quantifier-free first-order query in a database-independent way.
   Together, these three results yield the following picture:

$$\#\mathsf{qfCQ} \equiv \#\mathsf{qfFO} < \#\mathsf{CQ} < \#\mathsf{CQ}(\neq)$$

   Here, # stands for 'counts of', and $L_1 < L_2$ ($L_1 \equiv L_2$) means that for database-independent clients, a server answering queries in $L_2$ provides more (the same) information than a server answering queries in $L_1$.
4. Moving to adaptive strategies, we improve upon an earlier observation by Tyszkiewicz to note a general positive result: When the server answers boolean conjunctive queries with non-equalities, the client can derive the answer to *any* computable boolean query, using an adaptive strategy. Combining this result with the previous one, this means that for adaptive clients, a server answering just #qfCQ queries as above, already provides complete information about the database up to isomorphism.
5. Finally, we consider the case of a server answering boolean conjunctive queries (without non-equalities). We present a number of observations showing that the exact power of adaptive clients in this case is quite intriguing.

The basic setting of our work can be traced back to the framework of reflective relational machines (RRMs) introduced by Abiteboul, Papadimitriou and Vianu [2]. Indeed, RRMs are exactly the right computational model for adaptive

clients communicating with a database server. The original model focused on arbitrary boolean first-order queries as the query language; our work restricts this to the popular class of conjunctive queries, and brings counts into the picture. Different restrictions have already been studied by Tyszkiewicz. First, he revisited the complexity properties of the original model in the case of existential queries [8]. Then, he considered RRMs using boolean $k$-variable non-recursive Datalog queries, and showed that they cannot compute all of $k$-variable Datalog. He also considered RRMs using boolean $k$-variable first-order queries ($\mathrm{FO}^k$) with modular counting, and showed that they cannot express all of $\mathrm{FO}^k$ with counting [9]. RRMs using boolean $\mathrm{FO}^k$ queries have also been studied by Turull-Torres [7], who noted that they can compute precisely all computable queries invariant under $\mathrm{FO}^k$-equivalence. In contrast, we will show here that RRMs using conjunctive queries can *not* compute all computable queries invariant under conjunctive query equivalence.

## 2 Preliminaries

We work in the relational database model [1,10,11].

A *query* is a function $Q$, mapping databases (over some fixed schema) to relations. We assume familiarity with the class of conjunctive queries, $\mathsf{CQ}$, possibly extended with safe negation (including non-equalities), $\mathsf{CQ}(\neg)$, or just with non-equalities, $\mathsf{CQ}(\neq)$. In a *quantifier-free* conjunctive query, $\mathsf{qfCQ}$, all the variables of the body are also in the head.

*Example 1.* Over a database schema with a binary relation $R$ and a unary relation $S$, the following are examples of a $\mathsf{CQ}$, $\mathsf{CQ}(\neq)$, $\mathsf{CQ}(\neg)$, and $\mathsf{qfCQ}$, respectively.

$$\begin{aligned}(x) &\leftarrow R(x,y), R(y,z)\\ (x) &\leftarrow R(x,y), R(y,z),\ x \neq y\\ (x) &\leftarrow R(x,y),\ \neg S(y)\\ (x,y,z) &\leftarrow R(x,y), R(y,z)\end{aligned}$$

## 3 Database-Independent Model

A *natural number query* is a function, mapping databases to natural numbers. With any normal query $Q$ we can associate the natural number query $\#Q$, called the *count* of $Q$, that maps a database $D$ to the cardinality of the relation $Q(D)$.

**Definition 1.** *Let $L$ be a class of natural number queries, and let $Q$ be another natural number query. We say that $Q$* can be derived in a database-independent way from *$L$ if there is a finite sequence $Q_1, \ldots, Q_k$ of queries in $L$, and a computable function $f : \mathbb{N}^k \to \mathbb{N}$, such that*

$$Q(D) = f(Q_1(D), \ldots, Q_k(D))$$

*for any database $D$.*

*Example 2.* Let $L$ be $\#\mathsf{qfCQ}$: counts of quantifier-free conjunctive queries. Consider the following $\mathsf{qfCQ}(\neg)$ query:

$$Q(x, y) \leftarrow R(x, y), \neg S(y)$$

Then $\#Q$ can be derived in a database-independent way from $\#\mathsf{qfCQ}$, simply using the following two queries:

$$Q_1(x, y) \leftarrow R(x, y)$$
$$Q_2(x, y) \leftarrow R(x, y), S(y)$$

Indeed, we have $\#Q = \#Q_1 - \#Q_2$.

The technique illustrated in the above example can be generalized to arbitrary $\mathsf{qfCQ}(\neg)$ queries. Moreover, using the inclusion-exclusion principle $\#(A \cup B) = \#A + \#B - \#(A \cap B)$, we can generalize this further to *unions* of $\mathsf{qfCQ}(\neg)$ queries. We thus obtain:

**Proposition 1.** *The count of any union of* $\mathsf{qfCQ}(\neg)$ *queries can be derived in a database-independent way from* $\#\mathsf{qfCQ}$*; the derivation function is simply an integer arithmetic expression involving* $+$ *and* $-$*.*

The language of unions of $\mathsf{qfCQ}(\neg)$ queries can be thought of as the safe fragment of the quantifier-free first-order queries. For example, any relational algebra query involving equijoins, cartesian products, equality selections, unions, and differences, but not projections, is a union of $\mathsf{qfCQ}(\neg)$ queries. Things become simpler when we adopt the active-domain semantics for first-order queries (so that safety is no longer an issue), and are prepared to consider the query $(x) \leftarrow x = x$ a "conjunctive query". Then Proposition 1 can be reformulated by saying that *the count of any quantifier-free first-order query can be derived in a database-independent way from* $\#\mathsf{qfCQ}$.

We note the following variation of Proposition 1. An *injective* $\mathsf{qfCQ}$ query, denoted by $\mathsf{qfiCQ}$, is a $\mathsf{qfCQ}$ query with modified semantics to the effect that all variables must be instantiated by pairwise distinct values. Alternatively, $\mathsf{qfiCQ}$ can be viewed as the fragment of $\mathsf{qfCQ}(\neq)$ where non-equalities *must* be present between every pair of distinct variables. We have:

**Proposition 2.** *Proposition 1 also holds from* $\#\mathsf{qfiCQ}$ *instead of* $\#\mathsf{qfCQ}$*.*

This proposition actually follows from Proposition 1, because the count of any $\mathsf{qfCQ}$ query can be derived in a database-independent way from $\#\mathsf{qfiCQ}$, as illustrated in the following simple example.

*Example 3.* The count of $Q(x, y) \leftarrow R(x, y)$ can be derived using the queries $Q_1(x, y) \leftarrow R(x, y)$, $x \neq y$ and $Q_2(x) \leftarrow R(x, x)$. Indeed, $\#Q = \#Q_1 + \#Q_2$.

## 4 Quantification and Non-equality

So far, we have seen that in a quantifier-free world, things are pretty simple, and it does not matter whether the server answers merely conjunctive queries, or also supports negation. We will now see that things change when quantifiers come into play. We will first show that for database-independent clients, a server answering counts of arbitrary conjunctive queries truly provides more information than a server answering counts of quantifier-free ones only. We then show that adding even a single non-equality makes the server even more informative.

More concretely, for two classes $L_1$ and $L_2$ of natural number queries, we write $L_1 \leqslant L_2$ if any natural number query derivable in a database-independent way from $L_1$ is this also from $L_2$. We write $L_1 < L_2$ if $L_1 \leqslant L_2$ but not $L_2 \leqslant L_1$. Then we have:

**Theorem 1.** $\#\mathsf{qfCQ} < \#\mathsf{CQ}$.

**Theorem 2.** $\#\mathsf{CQ} < \#\mathsf{CQ}(\neq)$.

We will prove these theorems using the following two queries:

$$Q_1(x) \leftarrow S(x), R(x,y)$$
$$Q_2(x) \leftarrow R(x,x), R(x,y),\ x \neq y$$

Specifically, we will show that $\#Q_1$ is not derivable from $\#\mathsf{qfCQ}$, and $\#Q_2$ not from $\#\mathsf{CQ}$, in a database-independent way.

For any natural number $n$, define $G_n$ to be the structure over $\{R, S\}$ given by $S = \{0\}$ and $R = \{(0,1), (0,2), \ldots, (0,n)\}$. Any structure isomorphic to a $G_n$ is called a *star*. A database that is a disjoint union of stars is called a *constellation*.

For any natural number $n > 0$, define the following constellations. The constellation $A_n$ consists of $\binom{n}{i}$ copies of $G_i$ for every even $i \leq n$; the constellation $B_n$ consists of $\binom{n}{i}$ copies of $G_i$ for every odd $i \leq n$. We note:

**Lemma 1.** $\#Q_1(A_n) \neq \#Q_1(B_n)$.

*Proof.* In any constellation, $\#Q_1$ counts the number of non-$G_0$'s. Now $A_n$ contains one $G_0$, while $B_n$ contains no $G_0$'s. However, since $\sum_{i \text{ even}} \binom{n}{i} = \sum_{i \text{ odd}} \binom{n}{i}$, the total number of disjoint stars in $A_n$ is the same as in $B_n$. Hence the lemma follows.

Now suppose, for the sake of contradiction, that $\#Q_1$ is derivable in a database-independent way from $\mathsf{qfCQ}$. Since, by multiplication, counts of $\mathsf{qfCQ}$'s with a disconnected body are easily derivable in a database-independent way from counts of $\mathsf{qfCQ}$'s with a connected body, we may assume that the $\mathsf{qfCQ}$ queries used to derive $\#Q_1$ all have a connected body. Moreover, by Proposition 2, we can actually use $\mathsf{qfiCQ}$ queries. Let $\mathcal{C}$ be the finite set of connected $\mathsf{qfiCQ}$ queries used to derive $\#Q_1$. Let $n$ be the maximum number of variables used in a query in $\mathcal{C}$. Now observe that the result of applying a connected $\mathsf{qfiCQ}$ query with $j$ variables to a constellation equals the set of all embedded $G_{j-1}$'s. The following lemma, proven by a combinatorial calculation, is therefore crucial:

**Lemma 2.** *For any $j < n$, the number of embedded $G_j$'s is the same in $A_n$ as in $B_n$.*

We have now arrived at the desired contradiction. Indeed, Lemma 2 tells us that $A_n$ and $B_n$ are indistinguishable by counts of queries in $\mathcal{C}$. Since $\#Q_1$ was supposed to be derived using exactly these counts, this implies $\#Q_1(A_n) = \#Q_1(B_n)$, which contradicts Lemma 1.

Theorem 1 is thus proven. We move next to the proof of Theorem 2, which is similar, but a bit more complicated due to the lack of a counterpart to Proposition 2 in the case where we have quantification.

First, we need to adapt the notion of "star". For any natural number $n$, define $H_n$ to be the relation $\{(0,0),(0,1),(0,2),\ldots,(0,n)\}$. Any binary relation isomorphic to a $H_n$ is called a *loop-star*. The element corresponding to 0 is called the *center* of the loop-star; the other elements are called *rays*. Any disjoint union of loop-stars is called a *loop-constellation*.

Similarly to $A_n$ and $B_n$ from above, we introduce, for any natural number $n > 0$, the loop-constellation $A'_n$, which consists of $\binom{n}{i}$ copies of $H_i$ for every even $i \leq n$; and $B'_n$, which consists of $\binom{n}{i}$ copies of $H_i$ for every odd $i \leq n$. We have the following analogue to Lemma 1:

**Lemma 3.** $\#Q_2(A'_n) \neq \#Q_2(B'_n)$.

Now suppose, for the sake of contradiction, that $\#Q_2$ is derivable in a database-independent way from $\mathsf{CQ}$. Since, using multiplication, counts of $\mathsf{CQ}$'s with a disconnected body are easily derivable in a database-independent way from counts of $\mathsf{CQ}$'s with a connected body, we may assume that the $\mathsf{CQ}$ queries used to derive $\#Q_2$ all have a connected body. Let $\mathcal{C}$ be the finite set of connected $\mathsf{CQ}$ queries used to derive $\#Q_2$. Let $n$ be one more than the maximum number of variables used in a query in $\mathcal{C}$. If we can show that for any $Q \in \mathcal{C}$, we have $\#Q(A'_n) = \#Q(B'_n)$, then, in view of Lemma 3, we have arrived at the desired contradiction and proven Theorem 2.

Take an arbitrary $Q \in \mathcal{C}$, and an arbitrary loop-constellation $D$. Let $m$ be the number of variables in $Q$. For any $j \leq m$, define

$$S(j,Q,D) := \{t \in Q(D) \mid t \text{ contains exactly } j \text{ rays}\}.$$

Clearly,

$$\#Q(D) = \sum_{j=0}^{m} \#S(j,Q,D).$$

We therefore need to understand these sets $S(j,Q,D)$ better.

Fix $j$ arbitrarily. Let $\mathcal{E}$ be the set of all embedded images of $H_j$ in $D$, and let $\varepsilon(j,D)$ be the total number of these. For each $E \in \mathcal{E}$, fix an arbitrary embedding $e_E : H_j \to E$. Now define

$$T(j,Q,D) := \{e_E(u) \mid E \in \mathcal{E} \text{ and } u \in S(j,Q,H_j)\}.$$

We claim:

**Lemma 4.** $S(j, Q, D) = T(j, Q, D)$.

*Proof.* The inclusion from right to left is clear. For the converse inclusion, take $t \in S(j, Q, D)$. Writing $Q$ as $head \leftarrow body$, this means that there is a homomorphism $\sigma$ from *body* to $D$ such that $t = \sigma(head)$. Since *body* is connected, the image of $\sigma$ is part of a loop-star $H$ in $D$. Since $t$ contains $j$ rays, $H$ has at least $j$ rays. Now let $E$ be the image in $\mathcal{E}$ having as rays exactly those of $t$. Then we can define the following homomorphism $\sigma'$ from *body* to $E$:

$$\sigma'(x) := \begin{cases} \sigma(x) & \text{if } \sigma(x) \text{ appears in } t; \\ \text{the center of } E & \text{otherwise.} \end{cases}$$

This is indeed a homomorphism, thanks to the self-loop at the center of $E$. Since $e_E^{-1} \circ \sigma'$ is then a homomorphism from *body* to $H_j$, the tuple $u := e_E^{-1}(\sigma'(head))$ is in $Q(H_j)$. Moreover, $u \in S(j, Q, H_j)$, since the number of rays in $\sigma'(head) = t$ equals $j$. Hence, since $e_E(u) = t$, we conclude that $t \in T(j, Q, D)$ as desired.

We can now complete the proof as follows. Clearly, $\#T(j, Q, D) = \varepsilon(j, D) \times \#S(j, Q, H_j)$. Hence,

$$\#Q(D) = \sum_{j=0}^{m} \varepsilon(j, D) \times \#S(j, Q, H_j).$$

We now note the following analogue to Lemma 2:

**Lemma 5.** *For any $j < n$, we have $\varepsilon(j, A'_n) = \varepsilon(j, B'_n)$.*

Since the factors $\varepsilon(j, D)$ in the above expression for $\#Q(D)$ are the only that depend on $D$, we conclude that $\#Q(A'_n) = \#Q(B'_n)$, as desired.

## 5 The Adaptive Model

We now move on to the adaptive scenario, where a client, by performing an arbitrarily complex computation during which he interacts with the database server through conjunctive queries, tries to compute the answer to a more complex query. The computational model for this is the reflective relational machine (RRM) [2]. A RRM is a kind of oracle turing machine [4]. There is no input tape; instead, there is a read-only *answer tape* and a write-only *query tape*. Whenever the machine enters some special query state, it receives, on the answer tape, the answer to the query currently written on the query tape.

In the original model, the allowed queries are boolean first-order queries. We consider RRMs with boolean conjunctive queries, or more generally, counts of conjunctive queries.

Our first observation contrasts the adaptive model to our previous database-independent model. While we saw in the previous section that a database-independent client that can ask only counts of qfCQ queries cannot even derive some very simple counting queries, we now have:

**Theorem 3.** *Every computable natural number query can be computed by a RRM using counts of* qfCQ *queries.*

Here, it is understood, as usual, that computable queries are invariant under database isomorphisms [1].

Theorem 3 is actually a corollary of the following:

**Proposition 3.** *A RRM using boolean* $\mathsf{CQ}(\neq)$ *queries can construct an isomorphic copy of the database.*

Theorem 3 follows from Proposition 3 because we can get the answer to a boolean $\mathsf{CQ}(\neq)$ query by looking at the count of its quantifier-free version. Counts of $\mathsf{qfCQ}(\neq)$ queries can then in turn be expressed as counts of qfCQ queries, by Proposition 1.

We give:

*Proof (of Proposition 3).* For simplicity of notation only, we work with a single binary relation $R$. The proof is an improvement of observations already made by Abiteboul, Papadimitriou and Vianu [2] and Tyszkiewicz [8].[1]

We begin by determining the cardinality of the active domain of the database. If we adopt the active-domain semantics for first-order queries, this is easy: for progressively larger values of $n$, we ask queries of the form

$$() \leftarrow \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j$$

until the answer becomes false. If, however, we want to work only with safe queries (as we have done so far in this paper), we can still do this: by adding atoms $R(x_i, y_i)$ we can determine the cardinality $\#\pi_1(R)$ of the first projection of $R$; similarly we can determine $\#\pi_2(R)$, and $\#(\pi_1(R) \cap \pi_2(R))$. The cardinality of the active domain is then $\#\pi_1(R) + \#\pi_2(R) - \#(\pi_1(R) \cap \pi_2(R))$.

Suppose we have determined the active domain to have $n$ elements. Given an order on the active domain, we can write $R$ as an adjacency matrix. There are at most $n!$ possible such adjacency matrices. The following query asks whether at least one of these matrices has a 1 in the first row, first column:

$$() \leftarrow R(x_1, x_1)$$

– If the answer is yes, we ask next whether, of the matrices that have a 1 in the first row, first column, there is one that has also a 1 in the first row, second column:

$$() \leftarrow R(x_1, x_1), R(x_1, x_2),\ x_1 \neq x_2$$

If the answer is yes, we know there is a matrix starting with 11. If no, we know there is a matrix starting with 10 (even more, all matrices starting with 1 continue with 0.)

[1] [2] used full FO, [8] used $\mathsf{CQ}(\neg)$, and both ignored safety.

– If the answer is no, we know all matrices have a 0 in the first row, first column, so now we ask whether there is one that has a 1 in the first row, second column:

$$() \leftarrow R(x_1, x_2),\ x_1 \neq x_2$$

if the answer is yes, we know there is a matrix starting with 01; if no, we know all matrices start with 00.

And so on. After a total of $n^2$ queries we have recovered a complete adjacency matrix.

*Remark 1.* In this paper, we have not considered constants in conjunctive queries. In the previous two sections, nothing much changes in the presence of constants. In the adaptive model of the present section, however, constants would allow us to recover the database *exactly*, not up to isomorphism, by first determining the size of the domain as above, then systematically enumerating all possible domains of that size until a match is found, and finally determining the actual tuples in the database. Of course, the number of queries required then becomes totally unpredictable, while the number of queries used in the above proof is polynomial.

Proposition 3 raises the question of what we can do without inequalities. An immediate limitation on the power of a RRM using boolean conjunctive queries, denoted RRM(CQ), is that two databases that are indistinguishable by boolean conjunctive queries will of course be indistinguishable by the RRM as well. Formally, we call two databases $A$ and $B$ (over the same schema) CQ-*equivalent* if there are homomorphisms $f : A \to B$ and $g : B \to A$. It is well known [1, 11] that $A$ and $B$ are CQ-equivalent if and only if they are indistinguishable by boolean conjunctive queries. As a consequence, any boolean or natural number query computable by an RRM(CQ) must be invariant under CQ-equivalence.

*Example 4.* The binary relations (directed graphs) $\{(1,2),(1,3),(2,4),(3,4)\}$ and $\{(5,6),(6,7)\}$ are CQ-equivalent. This shows that queries depending on precise cardinalities, or on precise in- or out-degrees, or on precise structural properties (such as "is the graph a tree?" or "is the graph a chain?"), are all not computable by a RRM(CQ).

It is natural to conjecture that RRM(CQ)'s can compute *precisely* the computable queries invariant under CQ-equivalence. We can disprove this, however. Consider the boolean query SIMPLECYCLE over binary relations (directed graphs) that asks whether the graph is CQ-equivalent to a simple cycle. We have:

**Proposition 4.** *SIMPLECYCLE is not computable by a RRM(*CQ*).*

*Proof.* Suppose there is such a RRM, call it $M$. Consider the simple cycle $A = \{(1,2),(2,1)\}$. $M$ accepts $A$; let $n$ be the maximum number of variables used in any query asked by $M$ during its run on $A$. Pick an odd number $p > n$. Define

the graph $A'$ as the disjoint union of $A$ with a simple cycle of length $p$. Now any boolean conjunctive query $Q$ using at most $n$ variables cannot distinguish $A$ and $A'$. Hence, $M$ will accept $A'$ as well, which is incorrect because $A'$ is not CQ-equivalent to a simple cycle.

We conclude by showing that nevertheless, some non-trivial structural queries are computable by a RRM(CQ). Consider the following two natural number queries on directed graphs: SHORTCYCLE, mapping $G$ to the length of the shortest cycle if $G$ is cyclic, and to 0 otherwise; and LONGCHAIN, mapping $G$ to the length of the longest chain if $G$ is acyclic, and to 0 otherwise. We have:

**Proposition 5.** *Natural number queries SHORTCYCLE and LONGCHAIN are computable by a RRM(CQ).*

*Proof.* We pose the following queries for progressively larger values of $n$:

$$Q^n_{\text{chain}}() \leftarrow R(x_1, x_2), \ldots, R(x_{n-1}, x_n)$$
$$Q^n_{\text{cycle}}() \leftarrow R(x_1, x_2), \ldots, R(x_{n-1}, x_n), R(x_n, x_1)$$

If the graph is cyclic, $Q^n_{\text{cycle}}$ will become true at some point, at which the current value of $n$ gives us the value of SHORTCYCLE; if the graph is acyclic, $Q^n_{\text{chain}}$ will become false at some point, at which the current value of $n$ gives us the value of LONGCHAIN.

## 6 Concluding Remarks

A natural open question is, following the notation from Section 4, whether $\#\text{CQ}(\neq) < \#\text{CQ}(\neg)$. More generally, the expressibility of counts of queries in one language using counts of queries in another language seems to give rise to many interesting and mathematically challenging questions, with clear links to finite model theory [6]. We have only scratched the surface.

Another natural open question, given the popularity of conjunctive queries, is to understand the exact power of RRM(CQ).

It could also be worthwhile to study complexity issues in database interrogation using conjunctive queries, such as bounds on the number of queries needed to achieve a certain goal, and the size of these queries. Results of this nature for non-conjunctive queries have already been presented by Abiteboul, Papadimitriou and Vianu [2] and Tyszkiewicz [8,9].

**Acknowledgment.** We thank Jerzy Tyszkiewicz and Andrzej Szalas for inspiring discussions.

## References


1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. S. Abiteboul, C.H. Papadimitriou, and V. Vianu. Reflective relational machines. *Information and Computation*, 143(2):110–136, 1998.
3. N.R. Adam and J.C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.
4. M. Davis. *Computability and Unsolvability*. McGraw-Hill, 1958.
5. D. Dobkin, A.K. Jones, and R.J. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, 1979.
6. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, second edition, 1999.
7. J.M. Turull Torres. Reflective relational machines working on homogeneous databases. In *Foundations of Information and Knowledge Systems*, volume 1762 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2000.
8. J. Tyszkiewicz. Queries and algorithms computable by polynomial time existential reflective machines. *Fundamenta Informaticae*, 32:91–105, 1997.
9. J. Tyszkiewicz. Computability by sequences of queries. *Fundamenta Informaticae*, 48:389–414, 2001.
10. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.
11. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.
12. J.D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

# On the Difficulty of Finding Optimal Relational Decompositions for XML Workloads: A Complexity Theoretic Perspective*

Rajasekar Krishnamurthy, Venkatesan T. Chakaravarthy, and Jeffrey F. Naughton

University of Wisconsin, Madison, WI 53706, USA,
{sekar,venkat,naughton}@cs.wisc.edu

**Abstract.** A key problem that arises in the context of storing XML documents in relational databases is that of finding an optimal relational decomposition for a given set of XML documents and a given set of XML queries over those documents. While there have been a number of ad hoc solutions proposed for this problem, to our knowledge this paper represents a first step toward formalizing the problem and studying its complexity. It turns out that to even define what one means by an optimal decomposition, one first needs to specify an algorithm to translate XML queries to relational queries, and a cost model to evaluate the quality of the resulting relational queries. By examining an interesting problem embedded in choosing a relational decomposition, we show that choices of different translation algorithms and cost models result in very different complexities for the resulting optimization problems. Our results suggest that, contrary to the trend in previous work, the eventual development of practical algorithms for finding relational decompositions for XML workloads will require judicious choices of cost models and translation algorithms, rather than an exclusive focus on the decomposition problem in isolation.

## 1 Introduction

In order to leverage existing investments in relational database technology, there has recently been considerable interest in storing XML documents in relational databases. This problem has two main parts to it: (i) Given an XML schema (and possibly a query workload and statistics), choose a good relational decomposition and (ii) Given the XML schema and the corresponding relational decomposition, translate XML queries to SQL over this relational schema. While the two problems have been studied independently [3,4,6,10,12], they are actually closely related. In this paper, we study the relationship between the two problems, namely, choosing a good relational decomposition and using a good query translation algorithm.

* Research supported in part by NSF grants CSA-9623632, ITR-0086002, CCR-9634665 and CCR-0208013

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 270–284, 2003.

We show, through experiments with a commercial RDBMS and a well-known XML benchmark, that there exist translation algorithms $T_1$ and $T_2$, and relational decompositions $D_1$ and $D_2$, such that with translation $T_1$ decomposition $D_1$ is better than $D_2$, while with translation $T_2$ decomposition $D_2$ is better than $D_1$. This implies that one cannot talk about the quality of a decomposition without discussing the query translation algorithm to be used.

Any algorithm that attempts to find the optimal relational decomposition will have some cost model for the resulting relational queries, since it needs some way of evaluating the quality of the decomposition. To talk about the difficulty of finding good relational decompositions, we need to be specific about the cost models used. To show that the choice for the cost model can have a profound impact on the complexity of finding the optimal relational decomposition, we introduce two simple cost metrics and explore the complexity of the problem with each in turn.

Describing and analyzing the interaction between decompositions and query translations in full generality is a daunting task that appears (at least to us) unlikely to be amenable to a clean formal analysis; accordingly, here we consider a subset of the problem that is constrained enough to be tractable yet rich enough to provide insight into the general problem. We specify this subset by identifying a subset of XML schemas and a restricted class of XML queries over these schemas. We also specify a class of decompositions that, while not fully general, covers a wide range of decompositions we have seen in literature. We identify an important subproblem that must be addressed when designing an XML-to-Relational decomposition, which we call the *Grouping* problem. The subset of schemas and queries we consider captures the crux of the interaction between the decompositions and query translation algorithms in the context of the *Grouping* problem. We then present three query translation algorithms, *NaiveTranslation*, *SingleScan* and *MultipleScan*. In this setting, we look at how the complexity of choosing a good solution varies as we choose different combinations of translation algorithms and cost models.

Finally, we describe the *CompleteGrouping* problem, where we no longer fix the query translation algorithm, so the goal is to find the best pair of (relational decomposition strategy, query translation algorithm). We analyze the complexity of this problem for the two cost metrics.

The rest of the paper is organized as follows. We first show in Section 2 how the relative performance of different decompositions varies as the translation algorithm is varied. We also present the two cost metrics used in this paper. We then present a formal model describing the various parameters in the problem in Section 3 and formally define the *Grouping* problem. We describe the set-system coloring problem in Section 4 and show how it corresponds to the *Grouping* problem for the family of instances we consider. We also prove some complexity results for set-system coloring. We discuss the complexity of the *Grouping* problem in Section 5 as the query translation algorithm and cost model are varied and present our conclusions.

## 2 Relational Decompositions and Query Translation Algorithms

In this section, using the XMark benchmark XML schema [11], we illustrate the interaction between relational decompositions and XML-query to relational-query translation algorithms. We then describe two cost metrics used to compare alternative decompositions.

### 2.1 Motivating Example

In this section, we show how the quality of a decomposition is dependent on the query translation algorithm used. A part of the XMark schema is presented in Figure 1. Consider the various *Item* elements and the corresponding *InCategory* elements that appear in the schema. The techniques proposed in existing literature [3,12] map these elements into relations in one of two ways. The first way is to create six Item relations, one for each occurrence of Item in the schema. The Item elements are stored in relations AfricaItem, AsiaItem and so on based on their parent element. Corresponding *InCategory* relations are also created. Let us call this the *fully partitioned* strategy. The second way is to create an *Item* relation and an *InCategory* relation and store all the Item elements and their categories in these relations respectively. Let us call this the *fully grouped* strategy. Informally, how we decide to group the Item and InCategory elements into one or more relations is the *Grouping* problem. Consider the path expression query $Q$ in Figure 2. Let us now look at how this query is translated into a relational query. Consider the following simple algorithm. Identify all paths in the schema that satisfy the query. For each path, generate a relational query by joining all relations appearing in this path. The final query is the union of the queries over all satisfying paths (six paths for $Q$). This query translation algorithm is presented in Section 3.2. The relational queries for the fully-partitioned and fully-grouped strategies, $XQ^1_{fp}$ and $XQ^1_{fg}$ respectively, are given below. Notice how the two queries are very similar and differ mainly in the relations involved.

$XQ^1_{fp}$:

```
select count(*)
from   Site S, AfricaItem I1,
       AfricaInCategory C1
where  S.id = I1.parent
       and I1.id = C1.parent
       and C1.category = 'cat1'
union all ... (6 queries)
```

$XQ^1_{fg}$:

```
select count(*)
from   Site S, Item I, InCategory C
where  S.id = I.parent
       and I.id = C.parent
       and I.region = 'africa'
       and C.category = 'cat1'
union all ... (6 queries)
```

We know that, using the fully grouped strategy, relation InCategory contains exactly the set of elements returned by the path expression "/Site/Regions//Item/InCategory". So, we can translate query $Q$ into a selection query on relation InCategory. In a similar fashion, we can simplify the relational query for the fully partitioned strategy as well. This kind of optimizations can be performed by using the XML schema and XML-to-Relational mapping

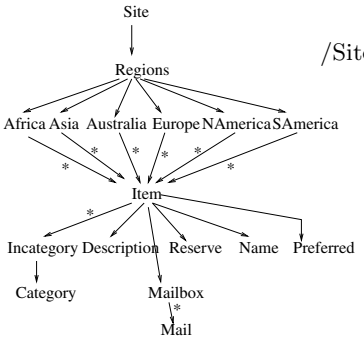



**Fig. 1.** XMark Benchmark schema

Find the number of items in a given category:
/Site/Regions//Item/InCategory[@Category = 'cat1']

**Fig. 2.** Path expression query $Q$

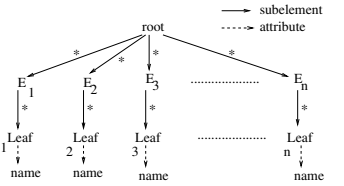



**Fig. 3.** Sample XML Schema in $\mathcal{T}$

information. Two such query translation algorithms are presented in Section 3.2 and 3.2. The resultant relational queries in this case, $XQ^2_{fp}$ and $XQ^2_{fg}$, are given below.

$XQ^2_{fp}$:

```
select count(*)
from   AfricaInCategory C1
where  C1.category = 'cat1'
union all ... (6 paths)
```

$XQ^2_{fg}$:

```
select count(*)
from   InCategory C
where  C.category = 'cat1'
```

On the 100MB XMark dataset [11], we noticed that $XQ^1_{fg}$ was 50% slower than $XQ^1_{fp}$, while $XQ^2_{fg}$ was about three times faster than $XQ^2_{fp}$. So, we see that for query $Q$, with algorithm *NaiveTranslation*, the *fully partitioned* strategy is better, whereas with algorithm *MultipleScan*, the *fully grouped* strategy is better. As a result, the quality of a decomposition is closely related to the query translation algorithm used.

### 2.2 Two Simple Cost Metrics

An algorithm that attempts to find the optimal relational decomposition needs a cost model for comparing alternative decompositions. One of our goals in this paper is to show that the choice of the cost metric has a big impact on the complexity of the problem. To illustrate this fact, we look at two simple cost models for relational queries.

- ***RelCount*:** The cost of a relational query is the number of relation instances in the relational algebra expression. This metric is inspired by historical work on minimizing unions of conjunctive queries [9].
- ***RelSize*:** The cost of a relational query is the sum of the number of tuples in relation instances in the relational algebra expression.

We next present some example scenarios where the above metrics are applicable. Consider the relational queries generated by algorithm *MultipleScan* for query $Q$.

The two queries, $XQ^2_{fg}$ and $XQ^2_{fp}$, have a selection predicate on the category attribute. If this predicate is a highly selective predicate and clustered indices are present on the category columns of these relations, then the relational optimizer will choose an index lookup plan for these queries. For query $XQ^2_{fg}$, the cost will be the sum of the cost of an index lookup and the cost for fetching all satisfying tuples. For query $XQ^2_{fp}$, it will be the sum of the cost of the six subqueries, each of which is the sum of the cost of an index lookup and the cost for fetching all satisfying tuples. Since, the query is highly selective, we can assume that the index lookup cost is the dominating cost and so $XQ^2_{fp}$ is six times more expensive than $XQ^2_{fg}$. In this scenario, *RelCount* is a good cost metric.

On the other hand, if the selection predicate is not very selective, then the optimizer may choose a plan that scans the relations. In this case, the cost of $XQ^2_{fg}$ depends on the size of the InCategory relation, while the cost of $XQ^2_{fp}$ depends on the sum of the sizes of the six region InCategory relations. In this scenario, *RelSize* is a good cost metric.

We would like to emphasize the fact that the purpose of these cost models is not to claim that they are optimal, or even very good in most cases, but rather that they are at least reasonable and they can be used to show how the complexity of the problem depends upon the chosen cost model.

## 3 Formal Model

To study the complexity of finding the optimal relational decomposition and how it depends on the query translation algorithm used and the cost model, we next formalize the problem in this section. We limit ourselves to a subset of the full problem by fixing the class of XML schemas and path expression queries that we consider. We then describe the class of relational decompositions, the class of relational queries that can be produced by the query translation algorithms and three candidate translation algorithms for the class of path expression queries we consider. Finally we define the *Grouping* problem, a subproblem lurking in the decomposition problem, which will be used to explore the complexity of the decomposition problem.

### 3.1 Definitions

**XML Schema:** We consider the schema $\mathcal{T}$, given in Figure 3. By varying $n$, the number of children of the root, we get a family of instances. Let *Label(v)* denote the name of the element $v$. If an edge is labeled with a "*", then the child element may occur more than once per parent element in the data. Otherwise the child element occurs exactly once per parent element. We refer to elements in the schema as *elements* and the corresponding elements in XML documents as *instance elements*. We refer to the element in $\mathcal{T}$ that corresponds to $/root/E_i/Leaf$ as element $Leaf_i$ or simply as $i$. Let Parent($Leaf_i$) denote the element $E_i$, which is the parent of $Leaf_i$. **Path expression queries:** We consider a class of path expression queries, $\mathcal{Q}$, that select a subset of the *Leaf* elements. A query $Q \in \mathcal{Q}$ is of the form $/\text{root}/(e_1|e_2|\ldots|e_k)/Leaf[@name = value]$, where $e_i$ is some $E_j$.

The query $Q$ selects a subset of the *Leaf* elements from the schema based on the path conditions in $Q$. We call this subset of *Leaf* elements *Range(Q)*. For example, for $Q = /\text{root}/(E_2|E_5|E_7)/Leaf[@name = value]$, $\text{Range}(Q) = \{2, 5, 7\}$. Applying the value conditions on the instance elements corresponding to elements in $\text{Range}(Q)$ will evaluate the result for $Q$.

**Relational Decompositions:** A number of relational decompositions have been proposed for XML data in literature [3,4,6,10,12]. These techniques can be broadly classified into two categories: (i) methods that use the XML schema information to decide on the relational schema [3,12] and (ii) methods that decide a relational schema independent of the XML schema [4,6,10]. For ease of exposition, we consider only the former techniques in this paper. We have extended Theorems 9-12 to the latter techniques as well.

As an example of decompositions based on the XML schema, for the schema in Figure 3, the *Shared* approach [12] will create $n + 2$ relations: one for storing *root* elements, one for each of the $n$ $E_i$ elements and one for storing all the *Leaf* elements (say relation $L$). The name attribute is stored in relation $L$ along with its parent *Leaf* element. The relation $L$ will have columns (*Leaf, name, parentid*) corresponding to the *Leaf* element, name attribute and information about the parent $E_i$ element.

In general, a relational decomposition for the XML schema can be viewed as a mapping function, $\sigma$, from vertices in $\mathcal{T}$ to a tuple $(R, C)$. The notation $\sigma(x) = (R_1, C_1)$ implies that instance elements of element $x$ are stored in column $C_1$ of relation $R_1$. Since $\sigma$ is a function, all instance elements of a single element are all stored in the same relation. Moreover, two elements in the XML schema can be mapped to the same column of a relation only if they have the same label. For example, $Leaf_1$ and $Leaf_2$ can be mapped to the same relational column, but $Leaf_1$ and the corresponding *name* attribute cannot be mapped to the same relational column.

**Relational Queries:** We consider the translation of the queries in $\mathcal{Q}$ to equivalent relational queries containing the select, project, join and union operators from relational algebra. These relational queries can also be viewed as the union of several conjunctive queries. As a technical detail, we do not allow disjunctions in selection and join conditions because our simple cost metrics do not capture the actual costs of relational queries in this scenario. This restriction can be lifted, but at the expense of requiring more expensive cost metrics and our results still hold. In this paper, for clarity of exposition, we restrict the class of operators allowed and use simple cost metrics. We also do not allow the set difference operator in the relational queries; the impact of lifting this restriction is an interesting topic for future work.

A Query $Q$ in $\mathcal{Q}$ selects one or more of the *Leaf* elements in $\mathcal{T}$. So, the equivalent relational query will be the union of one or more queries $Q_i$, where each $Q_i$ is a selection query or has a single join condition. If two *Leaf* elements $Leaf_i$ and $Leaf_j$ are mapped to the same relation $R$ and $\text{Range}(Q)$ contains just one of them (say $Leaf_i$), the relational query joins the relation $R$ with $\sigma(E_i)$ ($E_i$ is the parent of $Leaf_i$). For example, under the *fully grouped* decomposition

represented by mapping $\sigma$, the query $/root/E_1/Leaf$ will translate into the query $\sigma(E_1) \bowtie \sigma(Leaf_i)$.

The cost of a path expression query $Q$ for a given query translation algorithm is the cost of the relational query generated by the algorithm for $Q$.

The following definition will be useful in discussing later results.

**Definition 1.** *A set $S$ of elements in $\mathcal{T}$ is called a* scannable *set under a mapping $\sigma$ if all the elements in $S$ have the same label and are mapped to the same relational column $(R, c)$ and no other element $v \notin S$ is mapped to $(R, c)$.*

Intuitively if $S$ is a *scannable* set under $\sigma$, then a query $Q$ having $\text{Range}(Q) = S$ can be translated into a relational selection query on $R$.

The queries in $\mathcal{Q}$ have a selection condition on the *name* attribute of *Leaf* elements. So, it can be shown that storing the *Leaf* element and its *name* attribute in the same relation is better than storing them in two different relations. Similarly, it can be shown that decompositions that map two *Leaf* elements to two different columns of the same relation, have an equally good strategy that maps the *Leaf* elements to two different relations. This is due to the fact that disjunctions are not allowed in the relational queries. So, without loss of generality, we can assume that two *Leaf* elements are mapped to the same relation only if they are mapped to the same column[1].

We can also show that for the given family of instances $(\mathcal{T}, \mathcal{Q})$, under our cost metric, different mappings for the $E_i$ and *root* elements have the same performance. What really makes the difference is how the $n$ *Leaf* elements are stored in $k$ relations, for some $1 \leq k \leq n$.

The space of relational decompositions that needs to be examined to find the optimal decomposition consists of the various ways in which the *Leaf* elements can be grouped into $k$ relations. We next formally define the *Grouping* problem and the *CompleteGrouping* problem.

**Definition 2.** *Given a schema $\mathcal{T}$, a query workload $Q_W \subseteq \mathcal{Q}$ and an XML-to-relational query translation procedure, the* Grouping *problem is to find a partitioning of the $n$ Leaf elements into $k$ relations ($1 \leq k \leq n$) such that the sum of the cost of all queries in $Q_W$ is minimized.*

**Definition 3.** *Given a schema $\mathcal{T}$ and a query workload $Q_W \subseteq \mathcal{Q}$, the* CompleteGrouping *problem is to find the partitioning of the $n$ Leaf elements into $k$ relations ($1 \leq k \leq n$) and the XML-to-relational query translation such that the sum of the cost of all queries in $Q_W$ is minimized.*

### 3.2 Query Translation Algorithms

In this section, we briefly look at three different algorithms for translating a path expression query $Q$ into SQL. The *NaiveTranslation* algorithm was first presented in [12]. The other two algorithms are improvements over this algorithm for the class of path expression queries.

Let $\text{Range}(Q) = S = \{v_1, v_2, \ldots, v_m\}$.

[1] Notice that this automatically rules out solutions like a Universal relation for all the $n$ *Leaf* elements.

**Procedure** NAIVETRANSLATION($S$)

1 Let $\sigma(root) = (R_1, C_1)$.
  Set Result = *null*
2 For $i = 1$ to $m$
3  Let $\sigma(Parent(v_i)) = (R_2, C_2)$,
  $\sigma(v_i) = (R_3, C_3)$.
4  Result = Result $\cup$
  $\Pi_{C_3}(R1 \bowtie R_2 \bowtie R_3)$.

**Fig. 4.** *NaiveTranslation* algorithm

**Procedure** *SingleScan*($S$)

1 Set Result = *null*
2 If $(S, \sigma)$ is a *scannable* set, let $(R_1, C_1)$
  be the corresponding relational column
3  Result = $\Pi_{C_1}(R_1)$.
4 Otherwise,
5  For $i = 1$ to $m$
6  Let $\sigma(Parent(v_i)) = (R_2, C_2)$,
  and $\sigma(v_i) = (R_3, C_3)$.
7  Result = Result $\cup$ $\Pi_{C_3}(R_2 \bowtie R_3)$.

**Fig. 5.** *SingleScan* algorithm

**Procedure** *MultipleScan*(S)

1 Partition $S$ into equivalence classes $S_1, S_2, \ldots S_r$ based on the mapping $\sigma$.
2 Set Result = *null*
3 For $i = 1$ to r,
4  Let $S_i = \{v_1^i, v_2^i, \ldots, v_s^i\}$ and $\sigma(v_j^i) = (R_1, C_1)$.
5  If $(S_i, \sigma)$ is a *scannable* set,
6  Result = Result $\cup$ $\Pi_{C_1}(R_1)$.
7  Otherwise,
8  For $j = 1$ to $s$
9  Let $\sigma(Parent(v_j^i)) = (R_2, C_2)$, .
10  Result = Result $\cup$ $\Pi_{C_1}(R_1 \bowtie R_2)$.

**Fig. 6.** *MultipleScan* algorithm

***NaiveTranslation* Algorithm:** The naive translation strategy performs a join between the relations corresponding to all the elements appearing in a query. For example, let us consider the fully grouped strategy, where $\sigma(Leaf_i) = Leaf$. The query $/root/E_2/Leaf$ will be translated into a join among the *root*, $E_2$ and *Leaf* relations. A wild-card query will be converted into union of many queries, one for each satisfying wild-card substitution.

As an example, $//Leaf$ will be translated into the union of $n$ queries, one for each *Leaf* element. Each of these queries will be a join between the *root*, $E_i$ and $\sigma(Leaf_i)$ relations. Comparing this with the best translation scheme, which just performs a selection on the *Leaf* relation, we see that this scheme performs a lot of unnecessary computation. The translation procedure is presented in Figure 4. The condition on the attribute value is translated into a selection condition. This selection condition and the join conditions have been omitted in the above translation for clarity.

***SingleScan* Algorithm:** This translation procedure converts $Q$ into a selection query on a single relation, if Range($Q$) forms a *scannable* set. Otherwise, a separate relational query $Q_i$ is issued for each of the *Leaf* elements $Leaf_i$. $Q_i$

joins all relations on the schema path until the least common ancestor of all the *Leaf* relations has been reached. In this case, there is a join between the appropriate $E_i$ and $Leaf_i$ relations. For example, for the fully grouped strategy, $//Leaf$ will be a scan on the *Leaf* relation. But, $/root/(E_1|E_2|E_3)/Leaf$ will be translated as the union of three queries, one each for the three paths. The translation algorithm is given in Figure 5.

***MultipleScan* Algorithm:** Consider a query $Q$ with $\text{Range}(Q) = \{1, 2, 3, 4, 5\}$. Under a decomposition that groups *Leaf* elements $\{1, 2, 3\}$ and $\{4, 5, 6, \ldots, n\}$ into two different relations, the translation into a single selection will fail. So, under *SingleScan*, the relational query will be the union of 5 join queries. In *MultipleScan*, $\text{Range}(Q)$ is partitioned into equivalence classes, $\{1, 2, 3\}$ and $\{4, 5\}$, based on $\sigma$. Then, *SingleScan* is applied on each of these partitions and the resulting query is the union of these queries. In this case, $\{1, 2, 3\}$ gets translated into a single query, while $\{4, 5\}$ becomes the union of two queries and the translation of $Q$ is the union of these three queries. The translation algorithm is given in Figure 6.

## 4 Set System Coloring

In this section we define an abstract problem that is useful in investigating the complexity of the grouping problem. The problem of assigning relations to the $n$ *Leaf* elements can be viewed as assigning colors to the elements of a set of $n$ elements. We call this problem the *set system coloring* problem. We define four different cost metrics, $SSCost_{RC}$, $MSCost_{RC}$, $SSCost_{RS}$ and $MSCost_{RS}$ and show that this problem is NP-Hard and MAXSNP-Hard under the first three metrics. The problem is trivially solved under metric $MSCost_{RS}$. We also give a 2-approximation algorithm under metric $SSCost_{RC}$.

**Definition 4.** *A set-system $\mathcal{S}$ is 4-tuple $(A, \mathcal{H}, wt, mult)$, where $A = \{a_1, a_2, \ldots, a_n\}$ is a set, $\mathcal{H} \subseteq 2^A$ and $wt : A \longrightarrow \mathcal{Z}^+$ and $mult : \mathcal{H} \longrightarrow \mathcal{Z}^+$ are functions.*

A set-system can also be viewed as a hypergraph with weights on its vertices and hyperedges. So we call elements of $A$ *vertices* and those of $\mathcal{H}$ *hyperedges*. The function $wt$ assigns a positive integer weight to each vertex and $mult$ represents the multiplicity of the hyperedges. For a set $X \subseteq A$ let $wt(X) = \sum_{a \in X} wt(a)$.

The insight behind the above definition is the natural correspondence between the set-system problem and the *Grouping* problem defined in Section 3.1. Note that the vertices here correspond to the $n$ *Leaf* elements in the grouping problem and the hyperedges correspond to the queries in the workload. For a query $Q$, given a mapping $\sigma$, if $Leaf_i \in \text{Range}(Q)$ and $Leaf_i$ is not in a *scannable* set, then *MultipleScan* algorithm will output a join query for $leaf_i$. For metric *RelCount*, the number of joins performed in this query is captured by the weight on the vertex corresponding to $leaf_i$. Observe that the weight of each vertex is one for instances $(\mathcal{T}, \mathcal{Q})$. For metric *RelSize*, the size of the relations in the join query is captured by the $wt$ function. Here we assume that the input

also includes statistics about the cardinality of the elements corresponding to each schema node. *mult* represents the frequency of each query. Note that the query workload was defined as a set in Section 3.1 and the definition can easily be extended to include a frequency for each query in the workload.

**Definition 5.** *A coloring $\sigma$ is a mapping $\sigma : A \longrightarrow [1..n]$, where $n = |A|$. With respect to $\sigma$, a subset $X \subseteq A$ is said to be* scannable *if all the elements in $X$ have the same color and no element in $(A - X)$ has that color.*

The set-system coloring problem consists of finding a coloring of minimum cost. We can define a variety of optimization problems by supplying different cost metrics for computing the quality of a coloring for a set-system. For our results, four cost metrics are useful. We first define metrics $SSCost_{RC}$ and $SSCost_{RS}$. Then the other two metrics, $MSCost_{RC}$ and $MSCost_{RS}$, are defined in terms of these two metrics.

$SSCost_{RC}(\mathcal{S}, \sigma)$: Let $G$ be the set of *scannable* sets in $\mathcal{H}$, with respect to $\sigma$. Each set in $G$ contributes a cost of 1. For every set $h \in (\mathcal{H} - G)$, each element $a$ in $h$ contributes a cost of $wt(a) + 1$. Accounting for the multiplicities, we get $SSCost_{RC}(\mathcal{S}, \sigma) = \sum_{h \in G} mult(h) + \sum_{h \in (\mathcal{H}-G)} \sum_{a \in h} (wt(a) + 1) * mult(h)$.

$SSCost_{RS}(\mathcal{S}, \sigma)$: Let $G$ be the set of *scannable* sets in $\mathcal{H}$, with respect to $\sigma$. Each set in $G$ contributes a cost equal to its size. For every set $h \in (\mathcal{H} - G)$, each element $a$ in $h$ contributes a cost of $wt(a) + |\sigma(a)|$. Accounting for the multiplicities, we get
$SSCost_{RS}(\mathcal{S}, \sigma) = \sum_{h \in G} mult(h) * |h| + \sum_{h \in (\mathcal{H}-G)} \sum_{a \in h} (wt(a) + |\sigma(a)|) * mult(h)$.

$MSCost_{RC}(\mathcal{S}, \sigma)$ and $MSCost_{RC}(\mathcal{S}, \sigma)$: Partition each $h \in \mathcal{H}$ into $n$ equivalence classes $E(h) = \{h_1, h_2, \ldots, h_n\}$, where $h_i = \{a | a \in h \text{ and } \sigma(a) = i\}$. Let $\mathcal{S}_1$ denote the set system obtained by replacing each $h \in \mathcal{H}$ by the set $E(h)$. Then $MSCost_{RC}(\mathcal{S}, \sigma) = SSCost_{RC}(\mathcal{S}_1, \sigma)$ and $MSCost_{RS}(\mathcal{S}, \sigma) = SSCost_{RS}(\mathcal{S}_1, \sigma)$.

The above cost metrics give instances of the set-system coloring problem that correspond to different instances of the relational decomposition problem. For example, $SSCost_{RC}$ gives an instance of set-system coloring problem that corresponds to the problem of finding a relational decomposition for the schema in Figure 3 under query translation algorithm *SingleScan* and cost metric *RelCount*. Similarly, $MSCost_{RC}$, $SSCost_{RS}$ and $MSCost_{RS}$ correspond to the pairs (*MultipleScan*,*RelCount*), (*SingleScan*,*RelSize*) and (*MultipleScan*,*RelSize*) respectively.

We present the following results about the complexity of this problem under each of the above cost metrics.

### 4.1 Complexity Results under Metric $SSCost_{RC}$

In this section, we look at the complexity of set-system coloring under cost metric $SSCost_{RC}$.

**Theorem 1.** *Under metric $SSCost_{RC}$, set-system coloring is NP-Hard, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be $w$, for any constant $w \geq 0$.*

*Proof.* The problem of finding the vertex cover on 3-regular graphs is known to be NP-Complete [7]. We give a reduction from this problem.

Let $G = (V, E)$ be the input graph with $n$ vertices and $m$ edges. We output the set-system $\mathcal{S} = (A, \mathcal{H}, wt, mult)$. Let $A = E$ and $wt(a) = w$, for all $a$. For each $v \in V$, we add the set of (three) edges incident on $v$ to $\mathcal{H}$. The multiplicity of any hyperedge is one. Notice that $|\mathcal{H}| = n$. We can show that $G$ has a vertex cover of size $\leq k$ if and only if $\mathcal{S}$ has a coloring with cost $\leq n + (3w + 2) * k$.

Given a vertex cover $VC$ of $G$ of size $k$, we can construct a coloring $\sigma$ that uses $(n-k+1)$ colors. For each vertex $v \notin VC$, we color the three edges incident on $v$ by a unique color. For all the edges that are not yet colored, we use the $(n - k + 1)^{th}$ color. This coloring will have at least $(n - k)$ good hyperedges. Similarly, given a coloring with $(n - k)$ good hyperedges, the set obtained by adding the vertices corresponding to these hyperedges forms an independent set (of size $n - k$). Its complement is a vertex cover of size $k$. We have shown that the graph has a vertex cover of size $k$ if and only if the set system has a coloring with at least $(n - k)$ good sets. Moreover, if a coloring has $(n - k)$ good sets, then its cost is $n + (3w + 2) * k$. This completes the proof. □

As set-system coloring is NP-Hard, we explore the possibility of efficient approximation algorithms. We show that it can be approximated within a factor of 2. We also show that the problem is MAXSNP-Complete. Thus it cannot have a polynomial time approximation scheme (PTAS) unless NP=P.

**Theorem 2.** *For metric $SSCost_{RC}$, set-system coloring can be approximated within a factor of 2.*

*Proof.* We use the 2-approximation algorithm [2] for the weighted vertex cover problem (WVC). In WVC, given a graph with weights associated each vertex, the problem is to find the vertex cover of minimum cost. The cost of a vertex cover is the sum of the weights of the vertices in it.

Let the input set-system be $\mathcal{S} = (A, \mathcal{H}, wt, mult)$. First construct a weighted graph $G = (V, E)$ as follows. For each hyperedge $h \in \mathcal{H}$, add two nodes $v_h$ and $v'_h$ to the graph $G$ and add an edge between them. We call the node $v'_h$ as the companion of $v_h$. Add an edge between two nodes $v_s$ and $v_t$, if $s \cap t \neq \phi$. For each node $v_h$ set its weight to $(wt(h) + |h|) * mult(h)$. For the companion nodes, set the weight to be 1.

We next apply the known 2-approximation algorithm for WVC on $G$. Let the vertex cover returned by it be $VC$. We can construct a coloring $\sigma$ for $\mathcal{S}$ in the following manner. For each pair, $v_h, v'_h$ at least one of them has to be in $VC$. If both are present, remove $v'_h$ from $VC$. Now, for each node $v'_h$ in $VC$, give a unique color to all the elements in $h$. Assign a common new color to all the elements in $A$ that are not yet colored.

It can be shown that

$$cost(\sigma) \leq cost(VC) \leq 2 * cost(VC^*) \leq 2 * cost(\sigma^*)$$

where $\sigma^*$ is an optimal coloring of $\mathcal{S}$ and $VC^*$ is an optimal vertex cover of $G$.□

A problem is in MAXSNP if and only if it has some constant factor approximation algorithm [8]. So, by Theorem 2, the set-system coloring is in MAXSNP. The vertex cover problem on 3-regular graphs is known to be MAXSNP-Complete [1]. We can show that the reduction given in proof of Theorem 1 is actually an L-reduction. Thus we have the following theorem.

**Theorem 3.** *Under metric $SSCost_{RC}$, set-system coloring is MAXSNP-Complete, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be $w$, for any constant $w \geq 0$.*

**Theorem 4.** *Under metric $SSCost_{RC}$, set-system coloring cannot be approximated within a factor of 1.36 in polynomial time, unless NP=P.*

*Proof.* We can show that, for any constant $c > 1$, if there is a polynomial time algorithm that approximates the above coloring problem within a factor of $c$, then for any $\epsilon > 0$, we can approximate vertex cover within a factor of $c - \epsilon$. Using the inapproximability bound of 1.36 for vertex cover [5], we get the above result. □

### 4.2 Complexity Results for Other Cost Metrics

We have the following results for the other three cost metrics. We omit the proofs for lack of space.

**Theorem 5.** *Under metric $SSCost_{RS}$, set-system coloring is NP-Hard and MAXSNP-Complete, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be $w$, for any constant $w \geq 3$.*

**Theorem 6.** *Under metric $SSCost_{RS}$, when the weights of vertices are $<= 1$ and the multiplicities of the hyperedges are restricted to be one, set-system coloring is in P*

**Theorem 7.** *Under metric $MSCost_{RC}$, set-system coloring is NP-Hard and MAXSNP-Hard, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be $w$, for any constant $w \geq 0$.*

**Theorem 8.** *Under metric $MSCost_{RS}$, the coloring that assigns a different color for each vertex is an optimal solution.*

## 5 Complexity of the *Grouping* and *Completegrouping* Problems

Let us now look at the complexity of the *Grouping* problem for different query translation schemes under the two cost metrics.

**Theorem 9.** *When* NaiveTranslation *is the query translation algorithm,*

- *under metric* RelCount*, every relational decomposition is an optimal solution for the* Grouping *problem.*
- *under metric* RelSize*, the* fully partitioned *strategy is the optimal solution for the* Grouping *problem.*

**Proof Sketch:** Under the *NaiveTranslation* algorithm, for all decomposition strategies, the resultant relational query $Q$ is the union of the same number of queries. Each of these queries are similar in terms of the number of relations in it and they differ only in the actual relations. For metric *RelCount*, since the cost depends only on the number of relations, the cost is identical for all strategies. The relations involved in the query for the *fully partitioned* strategy are no larger than relations in any other strategy. So, for metric *RelSize*, the *fully partitioned* strategy is optimal. □

In Section 4 we showed that the set system coloring problem is NP-Hard and MAXSNP-Hard, under the two cost metrics, $SSCost_{RC}$ and $MSCost_{RC}$. We saw how the set-system coloring problem corresponds to the *Grouping* problem for the schema $\mathcal{T}$ and query workload $\mathcal{Q}$, when the elements to be grouped are represented as the set elements and the queries are represented as hyperedges. The weights of the vertices are set to one and multiplicity of hyperedges are set to the relative frequency of each query. It can be seen that, for the cost metric *RelCount*, the two metrics $SSCost_{RC}$ and $MSCost_{RC}$ for set-system coloring correspond to costs under the translation schemes *SingleScan* and *MultipleScan*. Hence, we have the following theorem.

**Theorem 10.** *Under metric* RelCount*, for translation algorithms* SingleScan *and* MultipleScan *finding the optimal solution for the* Grouping *problem is NP-Hard. The problem of grouping the nodes is MAXSNP-Hard.*

**Corollary 1.** *Under metric* RelCount*, the* Grouping *problem is NP-Hard and MAXSNP-Hard.*

The approximation algorithm for $SSCost_{RC}$ in Theorem 2 gives a 2-approximation algorithm for the *Grouping* problem under *SingleScan*. Even though we restricted our discussion in this paper to the family of schema $\mathcal{T}$, the grouping problem on a general XML schema can be represented as a set-system by setting the weights on the vertices of the set-system appropriately. The results for the set-system coloring will extend to these instances as well.

In a similar fashion, for metric *RelSize*, the results for set system coloring under metrics $SSCost_{RS}$ and $MSCost_{RS}$ carry over for the two translation algorithms and we have the following results.

**Theorem 11.** *Under metric* RelSize*, for translation algorithm* SingleScan *finding the optimal solution for the* Grouping *problem is NP-Hard. The problem of grouping the nodes is MAXSNP-Hard.*

**Table 1.** Complexity of *Grouping* problem

| | *NaiveTranslation* | *SingleScan* | *MultipleScan* |
|---|---|---|---|
| *RelCount* | any solution optimal | MaxSNP-Complete | MaxSNP-Hard |
| *RelSize* | fully partitioned | MaxSNP-Hard | fully partitioned |

**Theorem 12.** *Under metric* RelSize*, when* MultipleScan *is the query translation algorithm, the* fully partitioned *strategy is the optimal solution for the* Grouping *problem.*

It can be shown that *MultipleScan* is the optimal translation algorithm for the instances $(\mathcal{T}, \mathcal{Q})$. Hence, we have the following results.

LEMMA **1** *For the class of path expression queries $\mathcal{Q}$ for the schema $\mathcal{T}$,* MultipleScan *produces the relational query with minimum cost.*

**Theorem 13.** *Under metric* RelCount*, the* CompleteGrouping *problem is NP-Hard and MAXSNP-Hard.*

**Theorem 14.** *Under metric* RelSize*, the pair (fully partitioned strategy,*MultipleScan*) is an optimal solution to the* CompleteGrouping *problem.*

In summary, we see that the complexity of the *Grouping* problem depends a lot on the query translation algorithm used and the cost model that is appropriate for the given workload.

## 6 Conclusions and Future Work

In this paper, we considered the problem of finding optimal relational decompositions for XML workloads in a formal perspective. We identified the *Grouping* problem, an important sub-problem in choosing a good relational decomposition, and analyzed the complexity of this problem for three different query translation algorithms and two simple cost metrics. The results are summarized in Table 1. These results show that the query translation algorithm and the cost model play a vital role in the choice of a good decomposition, and choices for these dramatically affect the complexity of the problem.

Our work in this paper represents a first step toward formalizing and analyzing the problem of mapping XML data and queries to relational counterparts. Substantial room for future work exists in almost all directions. For example, it would be interesting to study the complexity of the problem for more elaborate cost functions, or relational queries that involve the difference operator, or more general classes of XML queries and schemas. It is our hope that the insight derived from our work here will be useful in such a study. Finally, we hope that the work presented here can serve as the basis for further work eventually leading to practical algorithms for choosing good relational decompositions for XML workloads.

## References


1. P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proc. 3rd Italian Conf. on Algorithms and Complexity, Lecture Notes in Computer Science, 1203*, pages 288–298. Springer-Verlag, 1997.
2. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.
3. P. Bohannon, J. Freire, P. Roy, and J. Simeon. From xml schema to relations: A cost-based approach to xml storage. In *ICDE*, 2002.
4. A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with stored. In *SIGMOD*, pages 431–442, 1999.
5. I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 33–42. ACM Press, 2002.
6. D. Florescu and D. Kossman. Storing and querying xml data using an rdbms. In *Data Engineering Bulletin*, volume 22, 1999.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
8. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
9. Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM (JACM)*, 27(4):633–655, 1980.
10. A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of xml documents. *Lecture Notes in Computer Science*, 1997, 2001.
11. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
12. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *Proceedings of the VLDB Conference*, 1999.


## A Horizontal Partitioning and the *Grouping* Problem

Though the *Grouping* problem may look very similar to the problem of horizontal partitioning of a relational table on a single disk, there are a few differences. While horizontal partitioning is usually done in physical database design, the *Grouping* problem deals with logical database design while automatically finding a good relational schema for the given XML schema. Moreover, in the XML context how we translate an XML query into a relational query plays an important role in the problem. For example, if we want only Africa entries from the InCategory relation, we will have to perform additional joins to get the results. The tree structure of an XML schema and presence of wild cards in XML queries also create situations where an arbitrary subset of the Item elements can be selected. On the other hand, in the horizontal partitioning context, the domain is usually ordered and queries either select a single partition or a range of partitions. So, the *Grouping* problem is actually a generalization of the horizontal partitioning problem. The latter corresponds to the *Grouping* problem when all the weights are 0. In this scenario, under metric *RelCount* the problem is MaxSNP-Hard, while under metric *RelSize* the problem is in $P$ for the three translation algorithms considered in this paper.

# Generating Relations from XML Documents*

Sara Cohen, Yaron Kanza, and Yehoshua Sagiv

School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
{sarina,yarok,sagiv}@cs.huji.ac.il

**Abstract.** This paper discusses several mechanisms for creating relations out of XML documents. A *relation generator* consists of two parts: (1) a tuple of path expressions *and* (2) an index indicating which path expressions may not be assigned the null value. Evaluating a relation generator involves finding tuples of nodes that satisfy the path expressions and are related to one another in a meaningful fashion. Different semantics for evaluation are given that take into account the possible presence of incomplete information. The complexity of generating relations from documents is analyzed and evaluation algorithms are described.

## 1 Introduction

Increasingly large amounts of data are accessible to the public in the form of XML documents. It is difficult for the naive user to query XML and thus, potentially useful information may not reach its audience. Search engines are currently the only efficient way to query the Web. These engines do not exploit the structure of documents and hence, are not well suited for querying XML.

We present several mechanisms for creating relations out of documents. The relations created can be used in many different ways. One use is to integrate our mechanisms into SQL in order to allow simultaneous querying of relations and XML. Another scenario where our mechanisms are useful is to create a universal relation interface to a set of documents, thereby enabling a simple and powerful search of the documents. It has been noted that the universal relation [8,11, 12] is a first step towards facilitating natural-language querying of relational databases. We believe that this also holds for XML documents.

Given a tuple of path expressions, we aim to find tuples of nodes from a given document that (1) match the path expressions *and* (2) are *meaningfully related*. We first try to decide when a pair of nodes are meaningfully related. In principle, any pair of nodes are related by virtue of being in the same document. However, as humans we can often determine that nodes are or are not meaningfully related by simply looking at a document. Several questions arise in this context:

- How can we automate the decision of whether a pair of nodes are related in a meaningful fashion? This becomes especially difficult when one considers the fact that documents may have varied structure.

* This work was supported by The Israel Science Foundation (Grant No. 96/01-1)

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 285–299, 2003.

– How can we deal with incompleteness in documents? If a document may be incomplete, then we may have to discover whether a particular node is meaningfully related to a node that does not even appear in the document.

Our mechanism for deciding whether a pair of nodes is meaningfully related attempts to capture our human intuition of relationships in a document.

Once we have determined which pairs of nodes are meaningfully related, we propose several different mechanisms that allow us to determine when larger tuples of nodes are meaningfully related. The more precise the mechanism, the less coverage it tends to have. Therefore, we present several mechanisms in order to allow the user to choose an appropriate one for a given domain. We also discuss the complexity of finding meaningfully related tuples, which varies depending on the mechanism used.

Section 2 presents some definitions and Section 3 presents our relation generating mechanisms. Some scenarios where our mechanisms are useful are described in Section 4. In Sections 5 we discuss the complexity of creating relations out of XML documents and present evaluation algorithms. Section 6 concludes.

## 2 Framework

**Relations.** A *tuple* has the form $t = (c_1 \colon a_1, \ldots, c_k \colon a_k)$ where $c_i$ and $a_i$ are a *column name* and a *value*, respectively. Tuples can have columns with null values, denoted $\perp$. We also allow tuples to have multiple columns with the same name. We call $(c_1, \ldots, c_k)$ the *signature* of $t$. A *relation* $\mathcal{R}$ is a bag of tuples with the same signature, also called the signature of $\mathcal{R}$.

**Trees.** We assume that there is a set $\mathcal{L}$ of labels and a set $\mathcal{A}$ of constants. An XML document is a tree $T$ in which each *interior node* is associated with a *label* from $\mathcal{L}$ and each *leaf node* is associated with *value* from $\mathcal{A}$. We denote the label of an interior node $n$ by *label*($n$) and the value of a leaf node $n'$ by *val*($n'$). We extend the *val* function to interior nodes $n$ by defining *val*($n$) to be the concatenation of the values of its leaf descendents. In Figure 1 there are three examples of such trees. In the sequel we will refer to these trees as $\mathcal{T}_{itm}$, $\mathcal{T}_{bk}$ and $\mathcal{T}_{athr}$. The nodes are numbered to allow easy reference.

Let $T$ be a tree and let $n_1$ and $n_2$ be nodes in $T$. Let $n$ be the *lowest common ancestor* of $n_1$ and $n_2$, and $T_n$ be the subtree of $T$ rooted at $n$. We denote by $T_{|n_1,n_2}$ the tree obtained by pruning from $T_n$ all nodes other than $n_1$ and $n_2$ that are not ancestors of either $n_1$ or $n_2$. We call this tree the *relationship tree* of $n_1$ and $n_2$. For example in $\mathcal{T}_{bk}$, the lowest common ancestor of 8 and 13 is 7. The relationship tree of 8 and 13 is comprised of the nodes 7, 8, 10 and 13.

**Graphs and Interconnection Graphs.** In order to create a relation out of a tree, we first must be able to decide which pairs of nodes are related in a meaningful fashion in a given tree. The relationship tree comes to our aid for this purpose.

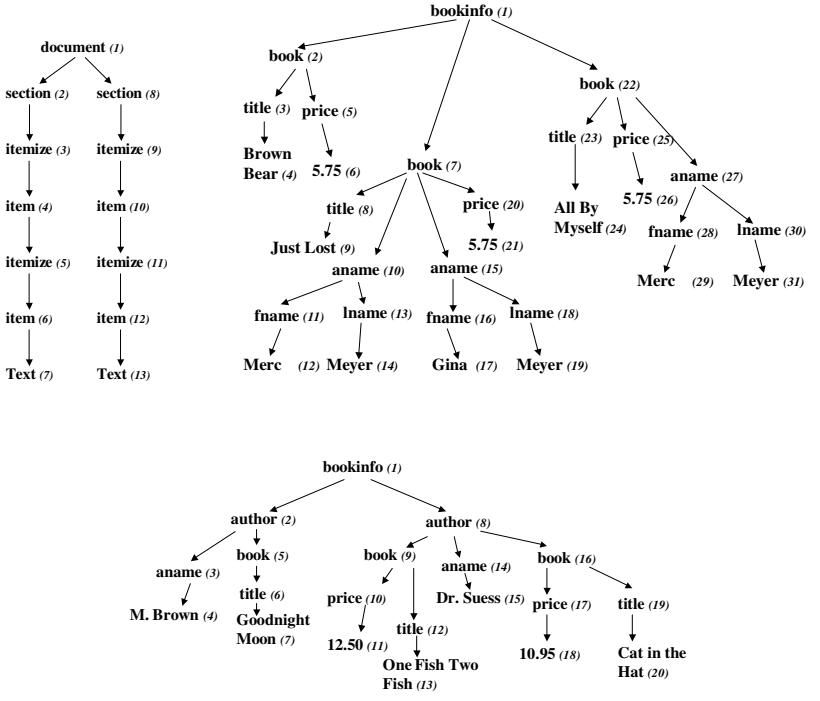



**Fig. 1.** Document $\mathcal{T}_{itm}$: itemize hierarchy (top left), $\mathcal{T}_{bk}$: bibliography grouped by book (top right), and $\mathcal{T}_{athr}$: bibliography grouped by author (bottom)

We start by giving an intuitive understanding of relationships in a document tree. Intuitively, a node in a tree represents an entity in the world. Two different nodes with the same label correspond to different entities of the same type. If $n_a$ is an ancestor of $n$, then we may understand that $n$ belongs to the entity that $n_a$ represents. Now, suppose that nodes $n$ and $n'$ have distinct ancestors $n_a$ and $n'_a$, respectively, such that $n_a$ and $n'_a$ have the same label. Suppose also that $n'_a$ is not an ancestor of $n$ and $n_a$ is not an ancestor of $n'$. We may conclude that $n$ and $n'$ are not meaningfully related since they belong to different entities of the same type. Note that $n_a$ and $n'_a$ must be in the relationship tree of $n$ and $n'$. Otherwise, they would be ancestors of both $n$ and $n'$ and would not imply that the nodes $n$ and $n'$ are not related.

We demonstrate and extend this intuition with a few examples. A formal definition of when nodes are related will be given later. Consider nodes 3 and 5 in $\mathcal{T}_{bk}$ (Figure 1). Their relationship tree does not contain two nodes with the same label. Therefore, nodes 3 and 5 are related. However, nodes 3 and 20 are not related since their relationship tree contains different nodes with the label book. This reflects the intuition that 3 is the title of the book with price node

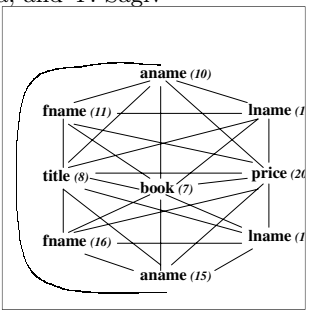



**Fig. 2.** The interconnection graph $\mathcal{IG}(\mathcal{T}_{bk}, \{7, 8, 10, 11, 13, 15, 16, 18, 20\})$

5 and not the title of the book with price node 20. Now, consider nodes 11 and 15 in $\mathcal{T}_{bk}$. Node 11 belongs to the aname node numbered 10. However, 15 is a different aname node. We may conclude that the fname in node 11 belongs to node 10 and is not related to node 15. Finally, consider nodes 10 and 15. These nodes share the same label. However, all their ancestors are the same, and thus, they belong to the same entities. Therefore, we may conclude that nodes 10 and 15 are meaningfully related. In fact, nodes 10 and 15 represent different author names, but they are related by virtue of belonging to the same book.

We formalize this idea. Let $n$ and $n'$ be nodes in $T$. We say that $n$ and $n'$ are *interconnected* if one of the following conditions holds:

- the relationship tree of $n$ and $n'$, i.e., $T_{|n,n'}$, does not contain two distinct nodes with the same label *or*
- the relationship tree of $n$ and $n'$, contains exactly one pair of distinct nodes with the same label and this pair is comprised of $n$ and $n'$.

Given a tree $T$, we define the *interconnection graph* of $T$, denoted $\mathcal{IG}(T)$. This undirected graph has the same set of nodes as those in $T$. There is an edge between nodes $n$ and $n'$ if $n$ and $n'$ are interconnected. Given a tree $T$ and a set of nodes $n_1, \ldots, n_k$, we denote the induced subgraph of $\mathcal{IG}(T)$, that contains the nodes $n_1, \ldots, n_k$, by $\mathcal{IG}(T, \{n_1, \ldots, n_k\})$. To illustrate these definitions, the interconnection graph $\mathcal{IG}(\mathcal{T}_{bk}, \{7, 8, 10, 11, 13, 15, 16, 18, 20\})$ is presented in Figure 2. Self loops have been omitted in the illustration.

We will be interested in interconnection graphs that have certain properties. A graph is *complete* if it contains an edge between every two nodes. A graph is *connected* if there is a path between every two nodes. Finally, a graph is a *star* if there is some node $n$ that is connected by an edge to every other node in the graph. Note that every complete graph is also a star graph and every star graph is also connected.

## 3 Creating Relations from Trees

An *atomic path expression*, denoted $\alpha$, is either a nonempty disjunction $(l_1 \mid \cdots \mid l_k)$ of labels from $\mathcal{L}$ or the *wildcard* symbol $*$. A *path expression*, denoted

$\omega$, is either an atomic path expression, or of the form $\omega'/\alpha$ or $\omega'/\!/\alpha$ where $\omega'$ is a path expression and $\alpha$ is an atomic path expression. We define recursively when a node $n$ in a tree $T$ *matches* a path expression $\omega$, denoted $n \models \omega$:

- $\omega = (l_1 \mid \cdots \mid l_k)$, the root of $T$ is $n$ and $label(n) = l_i$ for some $i \leq k$;
- $\omega = *$ and the root of $T$ is $n$;
- $\omega = \omega'/\alpha$, the parent of $n$ matches $\omega'$ and $n \models \alpha$ in the subtree rooted at $n$;
- $\omega = \omega'/\!/\alpha$, there is an ancestor of $n$ that matches $\omega'$ and $n \models \alpha$ in the subtree rooted at $n$.

For example, the path expression bookinfo/$*$ matches any child of a root that has the label bookinfo. The path expression $*/\!/$book/(aname | price) matches any aname or price node in a tree that is a child of a book node, regardless of the root's label. Finally, $*/\!/*$ matches any node in any tree. Note that given a node $n$ and a path expression $\omega$, it is possible to determine in polynomial time if $n \models \omega$.

The language of path expressions can be extended without affecting complexity results in this paper, as long as the extensions allow polynomial verification.

Let $T$ be a tree and $(\omega_1, \ldots, \omega_m)$ be a tuple of path expressions. We are interested in finding tuples of nodes $(n_1, \ldots, n_m)$ from $T$ that satisfy:

1. For all $i \leq m$, the node $n_i$ matches the path expression $\omega_i$ *and*
2. The nodes $n_1, \ldots, n_m$ are *meaningfully related*.

Requirement 1 is easily verified. On the other hand, Requirement 2 is rather difficult to determine. In accordance with the intuition presented in Section 2, we use the interconnection graph of a tree as an aid in deciding whether a set of nodes are meaningfully related. The interconnection graph only contains information about pairs of nodes that are related. We present three different semantics that enable us to decide whether larger tuples of nodes are related. Later we will compare the semantics in terms of complexity and expressive power.

- **Completely-Interconnected:** We say that a set of nodes $N$ are *completely-interconnected* in a tree $T$, denoted $\approx_c\{N\}$, if the interconnection graph of $T$, projected on $N$, i.e., $\mathcal{IG}(T, N)$, is a complete graph. Intuitively this states that a set of nodes is meaningfully related if every pair of nodes in the set is meaningfully related.
- **Reachably-Interconnected:** We say that a set of nodes $N$ are *reachably-interconnected* in a tree $T$, denoted $\approx_r\{N\}$, if $\mathcal{IG}(T, N)$ is a connected graph. The intuition behind this notion is that meaningful relationships are transitive, i.e., if $n_1$ and $n_2$ are meaningfully related and so are $n_2$ and $n_3$, then $n_1$ and $n_3$ must also be meaningfully related.
- **Star-Interconnected:** We say that a set of nodes $N$ are *star-interconnected* in a tree $T$, denoted $\approx_s\{N\}$, if $\mathcal{IG}(T, N)$ is a star graph. Intuitively, a set of nodes are meaningfully related if all the nodes in the set are meaningfully related to the same node.

Let $\Omega = (\omega_1, \ldots, \omega_m)$ be an $m$-tuple of path expressions and $T$ be a tree with a set of nodes $N$. We say that a function $\mu$: $\{1, \ldots, m\} \rightarrow N \cup \{\perp\}$ is a

*matching* of $\Omega$ to $T$ if for all $i \leq m$, either $\mu(i) \models \omega_i$ or $\mu(i) = \perp$. We denote the set of nodes in the image of a matching $\mu$ by $Image(\mu)$. We use $Mat_T^c(\Omega)$ to denote the set of matchings $\mu$, such that the nodes in $Image(\mu)$ are *completely-interconnected*, i.e., $\approx_c\{Image(\mu)\}$. Similarly, $Mat_T^r(\Omega)$ is the set of matchings $\mu$, such that $\approx_r\{Image(\mu)\}$, and $Mat_T^s(\Omega)$ is the set of matchings $\mu$, such that $\approx_s\{Image(\mu)\}$. It is not difficult to see that for all trees $T$ and tuples of path expressions $\Omega$,

$$Mat_T^c(\Omega) \subseteq Mat_T^s(\Omega) \subseteq Mat_T^r(\Omega).$$

Our matchings can have null values. However, we are interested in matchings that have maximal information. A matching $\mu'$ *subsumes* $\mu$, denoted $\mu \sqsubseteq \mu'$, if $\mu'$ gives the same value for every index that is assigned a non-null value by $\mu$, i.e., for all $i \leq m$, either $\mu'(i) = \mu(i)$ or $\mu(i) = \perp$. Intuitively, if $\mu'$ subsumes $\mu$, then $\mu'$ contains more information than $\mu$. An interconnection assignment $\mu$ is *maximal* if for every matching $\mu'$, we have that $\mu \sqsubseteq \mu'$ implies that $\mu = \mu'$.

Even a maximal matching can map some of the path expressions in $\Omega$ to null values. At times we may be interested in deriving only matchings that give non-null values to specific path expressions in $\Omega$. We call the pair $\Delta = (\Omega, k)$, where $\Omega$ is an $m$-tuple of path expressions and $k \leq m$, a *relation generator*. For $\square \in \{c, r, s\}$, we use $MMat_T^{\square}(\Omega, k)$ to denote the set of maximal elements in $Mat_T^{\square}(\Omega)$ that do not map any of the *first k path expressions* to the null value.

A relation generator $\Delta$ can be used in order to create a relation out of parts of an XML document. Given $\Delta = (\Omega, k)$ and a tree $T$, we compute $MMat_T^{\square}(\Omega, k)$ with $\square$ chosen as desired. (See Examples 3.1, 3.2 and 3.3 below and complexity analysis in Section 5 for a comparison of the semantics.) We can create a relation with signature $\Omega$ out of $MMat_T^{\square}(\Omega, k)$, in the obvious way. Formally, for each matching $\mu$ in the set, our relation contains a tuple $t_\mu$ with value $\mu(i)$ in column $\omega_i$. Depending on the end purpose of the relation, we may sometimes want to apply the *val* function to the nodes in the relation in order to derive tuples of strings instead of tuples of node ids.

*Example 3.1.* The relation generator $((*/\!/\mathsf{aname}, */\!/\mathsf{title}, */\!/\mathsf{price}), 1)$ finds triples of author names, titles and prices, all belonging to the same book. Only triples where the author name is non-null will be returned. For all three semantics, the relations created for $\mathcal{T}_{bk}$ ($\mathcal{T}_{athr}$) are the same. The tables for $\mathcal{T}_{bk}$ and $\mathcal{T}_{athr}$ are depicted in Figure 3(a) and 3(b). In parenthesis we note the value of the tuple after the *val* function is applied.

Note that there is no tuple corresponding to the title Brown Bear (node 4 in $\mathcal{T}_{bk}$) since it has no interconnected aname node. Observe also that the correct triples were found for both documents, even though they have different hierarchies.

*Example 3.2.* Using the tuple of path expressions $\Omega = (*/\!/\mathsf{title}, */\!/\mathsf{aname/fname}, */\!/\mathsf{aname/fname})$ we can create a relation generator that finds titles with pairs of first names of authors that wrote them. The set $MMat_T^c(\Omega, 3)$ will only contain

| *//aname | *//title | *//price |
|---|---|---|
| 10 (Mercy Meyer) | 8 (Just Lost) | 20 (5.75) |
| 15 (Gina Meyer) | 8 (Just Lost) | 20 (5.75) |
| 27 (Mercy Meyer) | 23 (All By Myself) | 25 (5.75) |

(a) Evaluated over $\mathcal{T}_{bk}$

| *//aname | *//title | *//price |
|---|---|---|
| 3 (M. Brown) | 6 (Goodnight Moon) | $\perp$ |
| 14 (Dr. Suess) | 12 (One Fish Two Fish) | 10 (12.50) |
| 14 (Dr. Suess) | 19 (Cat in the Hat) | 17 (10.95) |

(b) Evaluated over $\mathcal{T}_{athr}$

**Fig. 3.** Tables for $((*/\!/\mathsf{aname}, */\!/\mathsf{title}, */\!/\mathsf{price}), 1)$

matchings $\mu$ for which $\mu(2) = \mu(3)$, To see this, observe that in the interconnection graph in Figure 2, there are no two different fname nodes that are interconnected. This reflects the intuition that two fname nodes are not related since they belong to different authors. However, the sets $MMat^r_T(\Omega, 3)$ and $MMat^s_T(\Omega, 3)$ will contain matchings for the title Just Lost with the two different first names Mercy and Gina. Intuitively, this can be understood since the pair of names is related by virtue of both belonging to authors of the same title.

Consider now the tuple of path expressions $\Omega' = (*/\!/\mathsf{title}, */\!/\mathsf{aname/fname}, *//\mathsf{aname/lname})$. The set $MMat^c_T(\Omega', 3)$ will only contain matchings corresponding to first and last names of the same author. The sets $MMat^r_T(\Omega', 3)$ and $MMat^s_T(\Omega', 3)$ will contain in addition matchings with first names and last names of different authors of the same book. As before, such nodes are related since they belong to authors who wrote the same book. However, the set $MMat^c_T(\Omega', 3)$ may be perceived as a more precise answer. Therefore, we may conclude that choosing among the different semantics involves a trade-off between precision and recall (coverage).

*Example 3.3.* Tree $\mathcal{T}_{itm}$ in Figure 1 depicts two list hierarchies, as in a LaTeX document. Consider the tuple $\Omega = (*/\!/\mathsf{itemize}, */\!/\mathsf{item}, */\!/\mathsf{itemize}, */\!/\mathsf{item})$. The sets $MMat^c_T(\Omega, 4)$ and $MMat^s_T(\Omega, 4)$ do not contain matchings in which all nodes are different. The set $MMat^r_T(\Omega, 4)$ contains matchings corresponding to the quadruples (3, 4, 5, 6) and (9, 10, 11, 12). It does not, however, contain any matching with nodes from both list hierarchies.

## 4 Example Uses for Relation Generating Mechanism

Our mechanism for creating relations out of trees can be utilized in many different ways. We present several examples to illustrate possible uses.

### 4.1 Querying Trees and Relations Using SQL

Data is generally stored in relations. However, XML has become the standard for data exchange. Therefore, it is common to have both relations and XML as data sources. Posing queries against both types of data sources simultaneously is difficult. Our mechanism for translating trees to relations suggests one possible

solution. We extend the FROM clause of SQL to allow on-the-fly creation of relations from trees. Specifically, we use the predicates Complete, Reachable and Star to create tuples of nodes that match the given path expressions and are completely-, reachably- or star-interconnected, respectively.

For example, suppose that we wish to query the document $\mathcal{T}_{bk}$ and a table UserRatings(title, user, rating) stored in a relational database. The following query finds titles of books with a rating of at least 8 and author 'Smith'.

```
SELECT  Book.title
FROM    Complete(T_bk, ('*//title','*//aname'),0) as Book(title, author),
        UserRatings
WHERE   Book.title = UserRatings.title and
        Book.author like '%Smith%' and rating ≥ 8
```

The relation Book is created from the set $MMat^c_{\mathcal{T}_{bk}}((*/\!/title, */\!/aname), 0)$ by creating a tuple $t_\mu$ out of every matching $\mu$ in the computed set, in the obvious way. Note how we queried both XML and relations seamlessly.

### 4.2 An XML Search Engine

Currently, search engines cannot be used to query XML. More and more XML pages are finding their way onto the Web. Thus, it is becoming increasingly important to be able to query both the data and the meta-data content of the XML pages on the Web. Using the mechanism that we describe in this paper, a simple search language can be defined. A search query could have the form

$$path\_expression_1 : search\_phrase_1 \quad \cdots \quad path\_expression_n : search\_phrase_n$$

where $search\_phrase_i$ is a word or a quoted phrase. In addition, we allow the plus symbol to preface a path expression. Such path expressions must be matched to non-null values. Path expressions without a plus could be matched to null value.

As an example, the following query searches for books with a title containing the word XML and either the author Smith or no specified author.

$$+ */\!/\mathsf{title} : \mathsf{XML} \qquad */\!/\mathsf{aname} : \mathsf{Smith}$$

This query could be evaluated under any one of the three semantics (complete-interconnection, reachable-interconnection or star-interconnection). Basically, we would compute $MMat^c_T((*/\!/\mathsf{title}, */\!/\mathsf{aname}), 1)$, (or $MMat^r_T(\dots)$ or $MMat^s_T(\dots)$), on the documents $T$ available. Then, only the created tuples that satisfy the search conditions, i.e., that contain the specified phrases, would be returned.

## 5 Complexity of Creating Relations from Trees

In this section, we discuss the complexity of computing the sets $MMat^\square_T(\Omega, k)$ for some $\square \in \{c, r, s\}$, relation generator $(\Omega, k)$ and tree $T$. The complexity of evaluation is likely to be one of the factors that influence the decision of which

semantics to employ for a specific purpose. We will use the measure of *input-output complexity*, an extension of *combined complexity*, when analyzing the complexity of generating relations under the different semantics. In combined complexity both the document and the query are part of the input. In input-output complexity, we analyze the complexity of a problem as a function of the input (i.e., query and document) and the output. The choice of this complexity measure is justified both because it is of greater theoretical interest and because queries and query results may be large. In general, we will be interested in the following questions.

- **Non-emptiness:** Is $MMat_T^{\square}(\Omega, k)$ non-empty?
- **Evaluation:** How can we compute $MMat_T^{\square}(\Omega, k)$ efficiently?

For $\square = s$, i.e., when star-interconnected nodes are desired, it is not difficult to see that a relation generator can always be computed in polynomial time.

**Theorem 5.1 (Evaluation).** *Let $(\Omega, k)$ be a relation generator and $T$ be a tree. Then $MMat_T^s(\Omega, k)$ can be computed in polynomial time under input-output complexity.*

We present complexity results for the problems of non-emptiness and of evaluation for $\square = c$ and $\square = r$ below.

### 5.1 Complete-Interconnections

We solve the problems above for $\square = c$, i.e., when only completely-interconnected nodes are desired. We first show that given a relation generator $(\Omega, k)$ and a tree $T$, determining whether $MMat_T^c(\Omega, k)$ is non-empty is an NP-complete problem.

**Theorem 5.2 (Non-emptiness).** *Let $T$ be a tree and let $(\Omega, k)$ be a relation generator. Determining whether $MMat_T^c(\Omega, k) \neq \emptyset$ is NP-complete.*

*Proof. (Sketch)* The proof is by a reduction from 3-SAT, and is omitted because of space limitations. □

Since determining non-emptiness is NP-complete, we do not consider the general problem of finding all query results. There are, however, several important cases in which query evaluation is polynomial under input-output complexity. The first such case is when every path expression can be assigned the value null, i.e., for relation generators of the form $(\Omega, 0)$.

**Theorem 5.3 (Evaluation (Case 1)).** *Let $\Omega$ be a tuple of path expressions and $T$ be a tree. The set $MMat_T^c(\Omega, 0)$ can be computed in polynomial time under input-output complexity.*

*Proof. (Sketch)* Basically, we build up matchings in $MMat_T^c(\Omega, 0)$ gradually. We start with the matching that maps all path expressions to the null value and then try to extend this matching in all possible ways. This must be done in a

CompleteInterconnections$((\omega_1, \ldots, \omega_m), T)$

1. $\mathcal{M} := \{\emptyset\}$
2. **for** $i := 1$ to $m$ **do**
3. $\quad \mathcal{M}' := \emptyset$
4. $\quad$ **for each** $n \in T$ such that $n \models \omega_i$ **do**
5. $\quad\quad$ **for each** $\mu \in \mathcal{M}$ **do**
6. $\quad\quad\quad \mu' := \{(i, n)\} \cup \{\, (j, n') \in \mu \mid n'$ and $n$ are interconnected $\}$
7. $\quad\quad\quad \mathcal{M}' := \mathcal{M}' \cup \{\mu'\}$
8. $\quad \mathcal{M} := \mathcal{M} \cup \mathcal{M}'$
9. $\quad$ Remove strictly subsumed matchings from $\mathcal{M}$
10. **return** $\mathcal{M}$

**Fig. 4.** Polynomial algorithm to compute $MMat_T^c(\Omega, 0)$ (Theorem 5.3)

careful fashion to make sure that we do not create many matchings that will subsequently be removed, because they are not maximal.

An algorithm for creating $MMat_T^c(\Omega, 0)$ is presented in Figure 4. In this procedure, we represent matchings as sets of pairs of indices and nodes. For each matching we only represent explicitly pairs for which the index is not mapped to the null value.

We collect matchings in the set $\mathcal{M}$. In Line 1, the matching that maps all path expressions to null is added. Then, we loop over the path expressions (Line 2) and collect in $\mathcal{M}'$ matchings that map the $i$-th path expression to a non-null value. This is done by looping over all nodes $n$ that satisfy the $i$-th path expression (Line 4) and over all matchings $\mu$ created thus far (Line 5). We try to extend $\mu$ with $n$; however, nodes that are not interconnected with $n$ must be left out in order to derive a set of completely-interconnected nodes (Line 6). Subsumed assignments are removed in Line 9.

A formal proof of correctness is omitted due to lack of space. □

Even when we do not allow some of the path expressions to be mapped to the null value, it may still be possible to evaluate a relation generator in polynomial time under input-output complexity. We first present some necessary definitions and then describe a case in which polynomial evaluation is possible.

We say that a tree $T$ is *recursive* if $T$ contains nodes $n$, $n'$ such that $n'$ is a strict descendent of $n$ and $label(n) = label(n')$. Otherwise, we say that $T$ is *non-recursive*.

Given a tree $T$ and a path expression $\omega$, we use $MatchingNodes(\omega, T)$ to denote the set of nodes in $T$ that match $\omega$ and $MatchingLabels(\omega, T)$ to denote the set of all labels of nodes in $MatchingNodes(\omega, T)$. We denote by $LabelsAbove(\omega, T)$ all the labels $l$, such that there is a node labeled $l$ in $T$ with a descendent $n \in MatchingNodes(\omega, T)$. Note that by definition, if $n \models \omega$, then

$label(n) \in LabelsAbove(\omega, T)$. We associate each non-recursive tree $T$ and path expression $\omega$ with a relation $\mathcal{R}_{\omega,T}$. This relation has a column for each label in $LabelsAbove(\omega, T)$. For each node $n \in MatchingNodes(\omega, T)$, the relation $\mathcal{R}_{\omega,T}$ has a tuple $t_n$. The value of $t_n$ in column $l$ is the node id of the ancestor of $n$ with label $l$, if such an ancestor exists. Otherwise, the value in column $l$ is null. Since $T$ is non-recursive, there is at most one ancestor with label $l$ for any given node $n$. Thus, the tuples are well-defined. For each tuple $t_n$ created from node $n$ in $\mathcal{R}_{\omega,T}$, the *originating* column of $t_n$, denoted $Originating(t_n)$, is the column in which $n$ appears in tuple $t_n$, i.e., the column $label(n)$.

The notion of creating a hypergraph out of a set of relations has been studied for query optimization [12]. We review this idea briefly here. Relations $\mathcal{R}_1, \ldots, \mathcal{R}_m$ give rise to a hypergraph $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ in the following fashion:

- $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ has a node for each column in $\mathcal{R}_1, \ldots, \mathcal{R}_m$;
- for each relation $\mathcal{R}_i$, there is a hyperedge in $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ containing the nodes in the signature of $\mathcal{R}_i$, i.e., all the columns appearing in $\mathcal{R}_i$.

A hyperedge $e$ is an *ear* if (1) $e$ is the only hyperedge in the hypergraph *or* (2) there is a hyperedge $e'$ such that all nodes in $e \setminus e'$ are only in edge $e$. We call the removal of $e$ from the hypergraph *ear removal*. The *GYO-reduction* of a hypergraph is the result of applying ear removals (combined with the removal of nodes that do not belong to any hyperedge) until there remains no ear in the hypergraph. It has been proven that the GYO-reduction of a hypergraph is unique, i.e., is not dependent on the order in which ears are removed. A hypergraph is *acyclic* if its GYO-reduction is an the empty hypergraph (see also [5,15]). It has been proven [14,1,13] that if $\mathcal{H}(R_1, \ldots, R_m)$ is acyclic, then the natural join of $R_1, \ldots, R_m$ can be computed in polynomial time under input-output complexity.

The natural join of two relations requires equality on shared columns. By definition, the null value is never equal to any other value. Therefore, if a tuple has the null value in a shared column, this tuple will be lost in the result of the join. We define the pseudo natural join that differs from the natural join on exactly this issue, i.e., on how null values are dealt with. When performing a pseudo natural join, null values are always equal to any other value. The new value for the shared column, however, will be the non-null value, if such a value exists. For example, the pseudo natural join of the relations $\{(a\colon 1, b\colon \bot, c\colon 3), (a\colon 2, b\colon 3, c\colon 3)\}$ and $\{(a\colon \bot, b\colon 2, d\colon 4), (a\colon 1, b\colon \bot, d\colon 5)\}$ is the relation $\{(a\colon 1, b\colon 2, c\colon 3, d\colon 4), (a\colon 1, b\colon \bot, c\colon 3, d\colon 5)\}$. The pseudo natural join of relations with an acyclic hypergraph can be computed in polynomial time, in a fashion similar to computing the natural join. Note that the pseudo natural join, unlike the outer join, does not keep tuples that do not match other tuples in the way specified above.

**Theorem 5.4 (Evaluation (Case 2)).** *Let $((\omega_1, \ldots, \omega_m), m)$ be a relation generator and $T$ be a tree. Then $MMat^c_T((\omega_1, \ldots, \omega_m), m)$ can be computed in polynomial time under input-output complexity if the following conditions hold:*

1. *$T$ is non-recursive,*
2. *$\mathcal{H}(\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T})$ is acyclic,* and
3. *for all $i, j \leq m$, the sets MatchingLabels$(\omega_i, T)$ and MatchingLabels$(\omega_j, T)$ are disjoint if $i \neq j$.*

*Proof. (Sketch)* Let $T$ be a non-recursive tree. We are searching for sets of nodes that are completely-interconnected. Suppose that $n$ and $n'$ are interconnected and that $n$ has an ancestor $n_l$ with label $l$. Then, in the tree $T$ the node $n'$ must either not have any ancestor with label $l$, or have $n_l$ as its ancestor. If the tree is non-recursive, then this condition is sufficient and necessary for two nodes to be interconnected. Formally, for a non-recursive tree $T$, nodes $n$ and $n'$ are interconnected if and only if the following conditions hold.

- For every ancestor $n_l$ of $n$, either $n_l$ is an ancestor of $n'$ or $n'$ has no ancestor with label *label*$(n_l)$.
- For every ancestor $n'_l$ of $n'$, either $n'_l$ is an ancestor of $n$ or $n$ has no ancestor with label *label*$(n'_l)$.

Now, suppose that $\omega$ is a path expression for which $n \models \omega$. Similarly, suppose that $n' \models \omega'$. It is not difficult to see that the tuples $t_n$ in $\mathcal{R}_{\omega,T}$ and $t_{n'}$ in $\mathcal{R}_{\omega',T}$ will create a tuple in the pseudo natural join of $\mathcal{R}_{\omega,T}$ and $\mathcal{R}_{\omega',T}$ if and only if $n$ and $n'$ are interconnected.

In order to compute the set $MMat^c_T((\omega_1, \ldots, \omega_m), m)$, we first compute the pseudo natural join $\mathcal{R}$ of $\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T}$. This can be done in polynomial time, since $\mathcal{H}(\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T})$ is acyclic. Now, consider a tuple $t$ in $\mathcal{R}$. Let $t_1, \ldots, t_m$ be the tuples in $\mathcal{R}_1, \ldots, \mathcal{R}_m$ that joined together to create $t$. Then, the projection of $t$ on the columns *Originating*$(t_1), \ldots,$ *Originating*$(t_m)$ is a tuple containing completely-interconnected nodes. Since the sets *MatchingLabels*$(\omega_i, T)$ and *MatchingLabels*$(\omega_j, T)$ are disjoint for all $i$ and $j$, this tuple contains exactly $m$ different nodes. In this fashion, we can create all the matchings out of the tuples in the pseudo natural join. □

Building on Theorems 5.3 and 5.4, we can prove the following theorem.

**Theorem 5.5 (Evaluation (General)).** *Let $((\omega_1, \ldots, \omega_m), k)$ be a relation generator and $T$ be a tree. Then $MMat^c_T((\omega_1, \ldots, \omega_m), k)$ can be computed in polynomial time under input-output complexity if the following conditions hold:*

1. *either $k = 0$ or $T$ is non-recursive,*
2. *$\mathcal{H}(R_{\omega_1,T}, \ldots, R_{\omega_k,T})$ is acyclic,* and
3. *for all $i, j \leq k$, either*
   - *MatchingLabels$(\omega_i, T)$ and MatchingLabels$(\omega_j, T)$ are disjoint,* or
   - *MatchingNodes$(\omega_i, T)$ = MatchingNodes$(\omega_j, T)$.*

### 5.2 Reachable-Interconnections

We solve the problems of non-emptiness and evaluation for $\square = r$, i.e., when only reachably-interconnected nodes are desired. We first show that given a relation

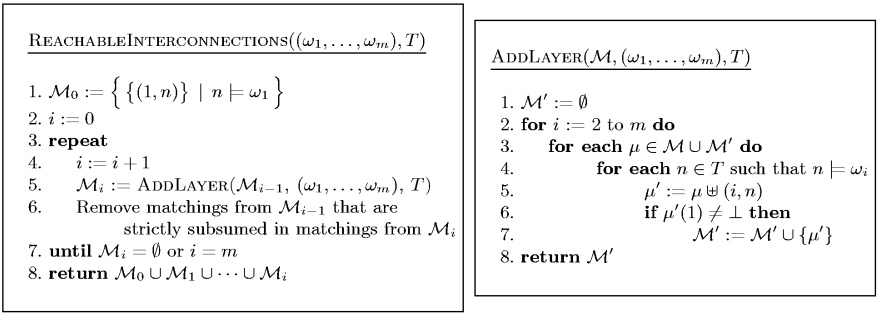



**Fig. 5.** Polynomial algorithm to compute $MMat^r_T(\Omega, 1)$ (Theorem 5.7)

generator $\Omega$, an integer $k$ and a tree $T$, determining whether $MMat^r_T(\Omega, k)$ is non-empty is an NP-complete problem. This is rather surprising since finding reachably-interconnected nodes only requires finding connected subgraphs in an interconnection graph.

**Theorem 5.6 (Non-emptiness).** *Let $T$ be a tree and let $(\Omega, k)$ be a relation generator. Determining whether $MMat^r_T(\Omega, k) \neq \emptyset$ is NP-complete.*

*Proof. (Sketch)* The proof is by a reduction from 3-SAT and is omitted because of space limitations. □

As in Section 5.1, we present an important case in which query evaluation is polynomial. If all path expressions can be assigned the null value, then the evaluation can be performed in polynomial time.

**Theorem 5.7 (Evaluation).** *Let $\Omega$ be a tuple of path expressions and $T$ be a tree. Then $MMat^r_T(\Omega, 0)$ can be computed in polynomial time under input-output complexity.*

*Proof. (Sketch)* Conceptually, creating matchings in $MMat^r_T(\Omega, 0)$ is more difficult than creating matchings in $MMat^c_T(\Omega, 0)$. The difficulty stems from the fact that nodes may be together in the image of a matching even if they are not directly interconnected, but rather connected by a path of nodes also in the image of the matching.

As in the proof of Theorem 5.3, we represent matchings as sets of pairs of indexes and nodes. We introduce some notation used in the algorithm. Given a matching $\mu$ and a set of nodes $N$ contained in the image of $\mu$, we define $\mathit{ConnectedSubMatching}(\mu, N)$ as the set of pairs $(i, n)$ in $\mu$, such that $n$ is connected to all nodes in $N$ in the graph $\mathcal{IG}(T, \mathit{Image}(\mu))$. Clearly, if $\approx_r\{\mathit{Image}(\mu)\}$, then $\mathit{ConnectedSubMatching}(\mu, N) = \mu$.

The operator $\updownarrow$ replaces the value for an index in a matching. The operator $\uplus$ performs an exchange and then removes non-connected nodes. Formally,

$$\mu \updownarrow (i, n) \stackrel{\text{def}}{=} \{ (j, n') \in \mu \mid j \neq i \} \cup \{ (i, n) \}$$

$$\mu \uplus (i, n) \stackrel{\text{def}}{=} \mathit{ConnectedSubMatching}(\mu \updownarrow (i, n), \{n\}).$$

In Figure 5 we present a polynomial algorithm for finding all matchings in $MMat_T^r(\Omega, 0)$ that do not assign $\omega_1$ the null value, i.e., the set $MMat_T^r(\Omega, 1)$. A complete proof of correctness is not given due to space limitations. In Line 1 of ReachableInterconnections, we create a matching for each node that matches $\omega_1$. While we succeed in extending a matching (Lines 3 and 7), we call the procedure AddLayer. There we loop over all the rest of the path expressions (Line 2). For each path expression, each matching $\mu$ created thus far, and each node $n$ matching the current path expression, we try to extend the $\mu$ with $n$ (Line 5). Only if the matching created still gives a non-null value to $\omega_1$, do we add this matching to the set of matchings created (Line 6). Since it is possible to find all matchings that do not assign $\omega_1$ a null value in polynomial time, we can repeat this process for each of the path expressions $\omega_i$ and thus, derive a polynomial algorithm for computing $MMat_T^r(\Omega, m)$. □

## 6 Conclusion

Relation generators, used for producing relations from XML documents were defined. Our relation generators allow naive users to retrieve interesting and naturally related portions of a document. Thus, they can be used for integrating relations and XML and as a foundation for XML search engines.

We presented the notion of an interconnection graph which describes connections between pairs of nodes. Several different semantics for finding larger tuples of related nodes from the interconnection graph were described. These semantics take into consideration that documents may not contain complete information, a situation that arises frequently in the context of the Web. Note that almost all our complexity results still hold even if the interconnection graph is created differently. For example, it is likely that the interconnection graph would be defined differently in the presence of either a schema or IDs and IDREFs. Once the interconnection graph was defined in this context, our mechanisms for creating tuples of meaningfully related nodes could be used.

This work extends [3]. Generating relations from semistructured data has been considered in the context of wrapper generation and inferring schemas from documents [4,6,9]. In [10], relations are created from semistructured data by schema generation. However, their approach is different as they attempt to "reverse-engineer" a website, while we simply look for semantic relationships between entities. Hence, the work in [10] is most applicable when the data has been constructed in a systematic manner, whereas our approach can be used even when the data does not conform to any schema. Interestingly, for many important special cases our complete answers coincide with their relations that are created using compact skeletons. In [2,7] a query language that uses a flexible semantics which can deal with variations in the data structure was presented. However, their approach was more restricted and their focus was on query equivalence.

One important open problem is how to define indices over an XML document that will allow relation generators to be quickly evaluated. Since some of the tuples produced by a relation generator may be more relevant than others, a

ranking system for such tuples should be defined. We intend to implement the mechanisms described here and to perform extensive experimentation in order to discover which semantics perform best and return the best results in practice.

## References


[1] U. Chakravarthy and J. Minker. Multiple query processing in deductive databases using query graphs. In *Proceedings of International Conference on Very Large Data Bases*, pages 384–391. Morgan Kaufmann, 1986.

[2] S. Cohen, Y. Kanza, and Y. Sagiv. SQL4X: A flexible query language for XML and relational databases. In *Proc. of the 8th International Workshop on Database and Programming Languages (DBPL)*, pages 263–280, Marino, (Rome, Italy), September 2001. Springer-Verlag.

[3] S. Cohen, Y. Kanza, and Y. Sagiv. Select project queries over xml documents. In *Proc. 5th Workshop on Next Generation Information Technologies and Systems*, pages 2–13, Caesarea (Israel), June 2002. Springer-Verlag.

[4] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from xml documents. In *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pages 165–176, Dallas (Texas, USA), May 2000. ACM Press.

[5] M. Graham. On the universal relation. Technical report, University of Toronto, Toronto (Canada), 1979.

[6] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. In *Proc. 14th International Conference on Data Engineering*, pages 297–301, Orlando (Florida, USA), Feb. 1998. IEEE Computer Society.

[7] Y. Kanza and S. Sagiv. Flexible queries over semistructured data. In *Proc. 20th Symposium on Principles of Database Systems*, pages 40–51, Santa Barbara (California, USA), May 2001. ACM Press.

[8] D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundation of the universal relation model. *ACM Trans. on Database System (TODS)*, 9(2):283–308, 1984.

[9] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data*, pages 295–306, Seattle (Washington, USA), June 1998. ACM Press.

[10] A. Rajaraman and J. D. Ullmann. Querying websites using compact skeletons. In *Proc. 20th Symposium on Principles of Database Systems*, pages 16–27, Santa Barbara (California, USA), May 2001. ACM Press.

[11] J. D. Ullman. The U. R. strikes back. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS)*, pages 10–22, Los Angeles, (California), March 1982. ACM Press.

[12] J. D. Ullman. *Principles of Database and Kowledge Base Systems*, volume II. Computer Science Press, 1989.

[13] E. Wong and K. Youssefi. Decomposition-a strategy for query processing. *ACM Trans. on Database Systems*, 1(3):223–241, 1976.

[14] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of International Conference on Very Large Data Bases*, pages 82–94. Morgan Kaufmann, 1981.

[15] C. Yu and M. Özsoyoglu. An algorithm for tree-query membership of a distributed query. In *Proceedings of IEEE COMPSAC*, pages 306–312, 1979.

# Containment for XPath Fragments under DTD Constraints

Peter T. Wood

Birkbeck College, University of London

**Abstract.** The containment and equivalence problems for various fragments of XPath have been studied by a number of authors. For some fragments, deciding containment (and even minimisation) has been shown to be in PTIME, while for minor extensions containment has been shown to be CONP-complete. When containment is with respect to trees satisfying a set of constraints (such as a schema or DTD), the problem seems to be more difficult. For example, containment under DTDs is CONP-complete for an XPath fragment denoted $\mathrm{XP}^{\{[]\}}$ for which containment is in PTIME. It is also undecidable for a larger class of XPath queries when the constraints are so-called simple XPath integrity constraints (SXICs). In this paper, we show that containment is decidable for an important fragment of XPath, denoted $\mathrm{XP}^{\{[],*,//\}}$, when the constraints are DTDs. We also identify XPath fragments for which containment under DTDs can be decided in PTIME.

## 1 Introduction

XPath [17] is a language for selecting nodes from XML document trees, and as such plays a crucial role in other XML technologies such as XSLT [18] and XQuery [19]. The expressions of XPath can be interpreted as simple queries on tree structures. The *containment problem* for XPath is to decide, given two XPath queries $Q_1$ and $Q_2$, whether $Q_1$ *contains* $Q_2$; that is, for every XML tree $t$, whether the output of $Q_1$ on $t$ contains the output of $Q_2$ on $t$. Since XPath query containment has many applications in XML querying, integration, transformation and active rule [2] environments, it has been the subject of much study recently [1,7,9,10,15,16].

Most of the papers cited above have studied different fragments of XPath, denoted $\mathrm{XP}^{\{[]\}}$, $\mathrm{XP}^{\{[],*\}}$, $\mathrm{XP}^{\{[],//\}}$ and $\mathrm{XP}^{\{[],*,//\}}$ in [9], depending on which XPath constructs are included in the fragment. All these fragments include node tests, composition of location steps ($/$), and predicates ($[\,]$). $\mathrm{XP}^{\{[],*\}}$ adds wildcards ($*$), $\mathrm{XP}^{\{[],//\}}$ adds the descendant axis ($//$), and $\mathrm{XP}^{\{[],*,//\}}$ includes both of these. Although not utilising the full capabilities of XPath, these fragments are commonly used to select nodes in XSLT, for example, and are interesting because of the relative complexity of the containment problem for them.

*Example 1.* The XPath query $a//b[*/i]/g$ selects nodes labelled with $g$ (which we call $g$-nodes for short) that are children of $b$-nodes, such that the $b$-nodes are both descendants of the root $a$-node and have an $i$-node as a grandchild.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 300–314, 2003.

It is also useful to be able to view a query in $\mathrm{XP}^{\{[\,],*,//\}}$ as a *tree pattern* [1, 9]. Fig. 1 gives the tree pattern for the query $a//b[*/i]/g$. In a tree pattern, double-lines indicate *descendant* edges (corresponding to //), while the return node is indicated in bold.

When using the descendant axis at the beginning of a predicate, we need to include "." (denoting the context node) in order to state that we want to search for descendants from that position in the query; without the "." searching would start from the root node. In fact, the XPath fragments we consider do not allow the second form of query. □

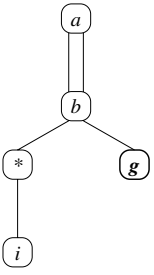



**Fig. 1.** The tree pattern for the XPath query $a//b[*/i]/g$.

Deciding containment for $\mathrm{XP}^{\{[\,],*\}}$ is in PTIME [16], a result which follows from the PTIME containment for acyclic conjunctive queries [20] (as pointed out in [9]). Containment for $\mathrm{XP}^{\{[\,],//\}}$ is also in PTIME [1]. However, as shown recently by Miklau and Suciu [9], when we extend the XPath fragment to $\mathrm{XP}^{\{[\,],*,//\}}$, containment becomes CONP-complete.

We are interested in studying containment in the presence of Document Type Definitions (DTDs). DTDs provide a means for typing, and therefore constraining, the structure of XML documents. Hence, while query $Q_1$ may not contain query $Q_2$ in general, it may be the case that, given a DTD $D$, $Q_1$ contains $Q_2$ when both are evaluated on documents valid for $D$.

*Example 2.* For the queries $Q_1 = a[b]/c$ and $Q_2 = a/c$, it is obvious that $Q_2 \supseteq Q_1$. The converse, however, does not hold since on a tree with only an $a$-node with $c$-node as a child, $Q_2$ returns the $c$-node while $Q_1$ returns the empty set. Now consider the following DTD $D$

$$
\begin{aligned}
a &\rightarrow (b, ((b, c)|d)) \\
b &\rightarrow ((e|f), (g|h)) \\
e &\rightarrow (i) \\
f &\rightarrow (i)
\end{aligned}
$$

which defines XML document trees that have a root node labelled $a$. The DTD $D$ states, for example, that every node labelled $a$ in a document tree valid for $D$

(or which *satisfies* $D$) must have a node labelled $b$ as a child. This is an example of what has been called a *child constraint* [15]. Such a constraint allows us to infer that $Q_1$ contains $Q_2$ under $D$, written $Q_1 \supseteq_{SAT(D)} Q_2$.

$D$ also requires that every $b$-node must have an $i$-node as a descendant. This means that $a[.//i]/c \supseteq_{SAT(D)} a/c$. As a final example, $D$ ensures that every path from an $a$-node to an $i$ node must pass through a $b$-node, which means that $a//b//i \supseteq_{SAT(D)} a//i$. □

As one might expect, when we consider containment of queries under constraints, the complexity increases. Given a DTD $D$, deciding containment under $D$ ($D$-*containment* for short), even for queries in $\mathrm{XP}^{\{[]\}}$, is CONP-complete [10, 16]. Containment is undecidable when the XPath fragment includes $\mathrm{XP}^{\{[],*,//\}}$ along with disjunction, variable binding and equality testing, and the constraints include so-called bounded simple XPath integrity constraints (SXICs) and those (unbounded) constraints implied by DTDs [7]. The decidability of containment of this XPath fragment in the presence of DTDs was stated as an open problem in [7]. In this paper, we prove that $D$-containment is decidable (EXPTIME-complete, in fact) for $\mathrm{XP}^{\{[],*,//\}}$. The same result is proved independently in [10].

The intractability of $D$-containment for $\mathrm{XP}^{\{[]\}}$ comes from the fact that inferring even some simple contraints from a DTD seems to be intractable [16]. As a result, we have been investigating classes of constraints which are implied by a DTD $D$ and which are both necessary and sufficient to determine $D$-containment for various classes of XPath queries in PTIME. For example, it was shown in [16] that so-called *sibling constraints* capture $D$-containment for queries in $\mathrm{XP}^{\{[]\}}$ that are *duplicate-free*. A query $Q$ in $\mathrm{XP}^{\{[]\}}$ is *duplicate-free* if each node in the tree pattern corresponding to $Q$ has children with distinct labels.

In this paper, we show that if DTD $D$ is *duplicate-free*, then $D$-containment for $\mathrm{XP}^{\{[]\}}$ is captured by sibling constraints and *functional constraints*, and can be decided in PTIME. A DTD $D$ is *duplicate-free* if each element name $n$ appears at most once in each content model (this does not preclude a node $n$ having multiple children with the same label in a tree satisfying $D$ since $n$ may still have a closure operator applied to it).

*Example 3.* Consider the following DTD $D$ which is duplicate-free:

$$a \rightarrow ((b*, c) \mid d)$$
$$b \rightarrow (e \mid f)$$
$$c \rightarrow (g?, h?)$$

$D$ does not imply any child constraints. However, it is the case that if an $a$-node has a $b$-node as a child, then it has a $c$-node as a child. This is an example of a *sibling constraint* (SC) [16], a generalisation of a child constraint, which allows us to show that $a[b][c] \supseteq_{SAT(D)} a[b]$.

It is also the case that $D$ requires that every $a$-node has at most one $c$-node as a child. This *functional constraint* (FC) allows us to show that $a/c[g] \supseteq_{SAT(D)} a[c/g]/c$. □

It is claimed in [16] that, if a DTD $D$ is duplicate-free, we can rewrite an $\text{XP}^{\{[]\}}$ query that is not duplicate-free as one that is duplicate-free while maintaining $D$-equivalence. However, the duplicate-free DTD $D$ of Example 3 and query $Q = a[b/e][b/f]$ show this is not true. $Q$ is $D$-satisfiable but there is no duplicate-free XPath query equivalent to $Q$. Hence, we need the results in the present paper to show PTIME containment under duplicate-free DTDs for $\text{XP}^{\{[]\}}$.

Finally, given the examples of constraints implied by a DTD in Example 2, one might wonder whether a richer class of constraints than SCs and FCs, but less powerful than DTDs themselves, might capture $D$-containment for $\text{XP}^{\{[]\}}$ when neither $D$ nor the queries are duplicate-free. We prove that this is not the case, in the sense that to capture $D$-containment for $\text{XP}^{\{[]\}}$ requires the ability to express exactly the *unordered* language generated by $D$.

In the next section, we introduce the necessary definitions and notation for XML trees, XPath queries and DTDs. In Section 3, we prove that containment under DTDs is decidable for queries in $\text{XP}^{\{[],*,//\}}$. We show this by transforming queries in $\text{XP}^{\{[],*,//\}}$ to regular tree grammars and using decidability and closure results for regular tree grammars, since DTDs are a special case of regular tree grammars. In Section 4, we show that, for queries in $\text{XP}^{\{[]\}}$, sibling and functional constraints are necessary and sufficient to decide containment under duplicate-free DTDs in PTIME. We prove in Section 5 that this result cannot be extended beyond duplicate-free DTDs. Section 6 contains a discussion of related work, while Section 7 concludes.

## 2 XML Trees, XPath Queries, and DTDs

Let $\Sigma$ be a finite alphabet of XML element names. Since the XPath fragments we consider cannot query attributes or the textual contents of nodes, we define a *document tree* (or simply *tree*) over $\Sigma$ to be an ordered, unranked finite structure with nodes labelled by element names from $\Sigma$. So all leaf nodes (elements) are assumed to be empty. The set of all trees over $\Sigma$ is denoted by $T_\Sigma$. Given a tree $t \in T_\Sigma$, the root of $t$ is denoted by $root(t)$, the nodes of $t$ by $nodes(t)$, and the label of node $x \in nodes(t)$ by $\lambda(x) \in \Sigma$.

Expressions in $\text{XP}^{\{[],*,//\}}$ (which we also loosely refer to as queries) are defined by the following grammar, in which $n$ denotes an element name and "." refers to the current node:

$$p \rightarrow p \text{ '/' } p \mid p \text{ '//' } p \mid p \text{ '[' } p \text{ ']' } \mid n \mid \text{'*'} \mid \text{'.'}$$

Expressions enclosed in '[' and ']' are called *predicates*. The use of the current node in expressions such as $a/./b$ or $a//.//b$, it can be eliminated. Expressions in $\text{XP}^{\{[]\}}$ do not use ".", "$*$" or "//".

Given a query $Q$ in $\text{XP}^{\{[],*,//\}}$ and a tree $t \in T_\Sigma$, $Q(t)$ denotes the set of nodes that is the result of evaluating $Q$ on $t$ according to the semantics given in [14]. We assume that the context node for evaluating $Q$ on $t$ is always $root(t)$. If $Q(t) \neq \emptyset$, we say that $t$ *satisfies* $Q$. The set of trees over $\Sigma$ satisfying $Q$ is denoted $SAT_\Sigma(Q)$. Let $P$ and $Q$ be queries in $\text{XP}^{\{[],*,//\}}$. We say that $P$

*contains* $Q$, written $P \supseteq Q$, if for all trees $t \in T_\Sigma$, $P(t) \supseteq Q(t)$. In addition, $P$ is *equivalent* to $Q$, written $P \equiv Q$, if $P \supseteq Q$ and $Q \supseteq P$.

Similar to [1,9], we consider queries also as *tree patterns*. A pattern $Q$ is an unordered tree over alphabet $\Sigma \cup \{*\}$ with a distinguished subset of edges called *descendant edges*, and a distinguished node called the *result node*. Edges that are not descendant edges are called *child edges*. An example of a tree pattern is given in Fig. 1, where child edges are represented by single lines, descendant edges by double lines, and the result node by a boldface circle and label. The result node of pattern $Q$ is denoted $result(Q)$. From now on, we use XPath queries and tree patterns interchangeably.

As in [9], an *embedding* or *homomorphism* from pattern $Q$ to tree $t$ is a mapping $h$ from the nodes of $Q$ to the nodes of $t$ such that (1) $h$ maps $root(Q)$ to the $root(t)$, (2) for each node $x$ in $Q$, $\lambda(x) = *$ or $\lambda(x) = \lambda(h(x))$, and (3) for each node $x$ and $y$ in $Q$, if $(x, y)$ is a child edge in $Q$ then $(h(x), h(y))$ is an edge in $t$, and if $(x, y)$ is a descendant edge in $Q$ then $h(y)$ is a proper descendant of $h(x)$ in $t$. If the result node of $Q$ is $x$, then

$$Q(t) = \{(h(x)) \mid h \text{ is a homomorphism from } Q \text{ to } t\}$$

$Q(t)$ is represented as a set of tuples rather than simply a set of elements, because we also want to allow Boolean patterns and to be able to distinguish between Boolean patterns that are satisfied by some tree and those that are not [9]. A *Boolean* pattern $Q$ is a pattern in which there is no result node: $Q(t)$ evaluates to the empty tuple if there a homomorphism from $Q$ to $t$; otherwise, $Q(t) = \emptyset$. When $Q$ is not a Boolean pattern, we will often consider $Q(t)$ to be simply a set of elements.

A containment mapping between queries is similar to a homomorphism from a query to a tree, the differences arising because queries have result nodes and descendant edges which trees do not. A *containment mapping* from query $Q_1$ to query $Q_2$ is a mapping $h$ from the nodes of $Q_1$ to the nodes of $Q_2$ such that (1) $h$ maps $root(Q_1)$ to $root(Q_2)$, (2) $h$ maps $result(Q_1)$ to $result(Q_2)$, (3) for each node $x$ in $Q_1$, $\lambda(x) = *$ or $\lambda(x) = \lambda(h(x))$, and (4) for each node $x$ and $y$ in $Q_1$, if $(x, y)$ is a child edge in $Q_1$ then $(h(x), h(y))$ is a child edge in $Q_2$, and if $(x, y)$ is a descendant edge in $Q_1$ then $(h(x), \ldots, h(y))$ is a path of child and/or descendant edges in $Q_2$.

The existence of a containment mapping is always a sufficient condition for containment between queries. It is also a necessary condition for containment for $\mathrm{XP}^{\{[],*\}}$ and $\mathrm{XP}^{\{[],//\}}$ (and, of course, for $\mathrm{XP}^{\{[]\}}$), but it is *not* a necessary condition for containment for $\mathrm{XP}^{\{[],*,//\}}$ [9].

A *document type definition* (DTD) $D$ over finite alphabet $\Sigma$ consists of a root type in $\Sigma$, denoted $root(D)$, and a mapping that associates with each $a \in \Sigma$ a regular expression over $\Sigma$. If the mapping associates with $a$ the regular expression $R^a$, then we say that $R$ is the *content model* for $a$ and write $a \to R^a$ (which we also call a *production*). By convention, we often simply write down the productions for $D$, with the first production being for the root type and with $\Sigma$ comprising all symbols which appear in the productions. For example, the root type of the DTD $D$ of Example 2 is taken as $a$ and $\Sigma = \{a, b, c, d, e, f, g, h, i\}$.

For regular expressions, we use the conventions from XML DTDs, namely that ',' denotes concatenation, '|' denotes alternation, and '$*$' denotes Kleene closure. We represent the regular expression denoting the empty string by $\epsilon$, so that, for regular expression $R$, $R^+$ is shorthand for $R, R^*$, and $(R)?$ is shorthand for $(R \mid \epsilon)$. If $R$ is a regular expression, then $L(R)$ is the *language* denoted by $R$.

A tree $t \in T_\Sigma$ *satisfies* a DTD $D$ over $\Sigma$ if $\lambda(root(t)) = root(D)$ and for each node $x$ in $t$ with sequence of children $y_1, \ldots, y_n$, the string $\lambda(y_1) \cdots \lambda(y_n)$ is in $L(R^{\lambda(x)})$ (the language for the content model of the label of $x$). The set of trees satisfying DTD $D$ is denoted $SAT(D)$.

Given XPath queries $Q_1$ and $Q_2$, $Q_1$ *$D$-contains* $Q_2$, written $Q_1 \supseteq_{SAT(D)} Q_2$, if, for every tree $t \in SAT(D)$, $Q_1(T) \supseteq Q_2(T)$. $Q_1$ and $Q_2$ are *$D$-equivalent*, written $Q_1 \equiv_{SAT(D)} Q_2$, if $Q_1 \supseteq_{SAT(D)} Q_2$ and $Q_2 \supseteq_{SAT(D)} Q_1$.

Let $t \in T_\Sigma$ be a (document) tree, $a, c \in \Sigma$ be element names, and $B \subseteq \Sigma$ be a set of element names. Tree $t$ *satisfies* the *sibling constraint* (SC) $a : B \Downarrow c$ if whenever a node labelled $a$ in $t$ has children labelled with each $b \in B$, it has a child node labelled with $c$ [16]. When $B = \emptyset$, the SC is called a *child constraint* [15]. An SC (over $\Sigma$) is *trivial* if it is satisfied by every tree $t \in T_\Sigma$. So $a : B \Downarrow c$ is trivial if $c \in B$. If $S$ is a set of SCs over $\Sigma$, then $SAT(S)$ denotes the set of trees in $T_\Sigma$ which satisfy each SC in $S$.

Let $s$ be the SC $a : B \Downarrow c$. Regular expression $R^a$ *implies* the SC $s$, written $R^a \models s$ if whenever a string $w \in L(R^a)$ contains each element of $B$, it also contains $c$. SC implication was shown to be CONP-hard in [16]. DTD $D$ *implies* the SC $s$, written $D \models s$, if $R^a$ implies $s$ or, equivalently, each $t \in SAT(D)$ satisfies $s$. $D$ *implies* the set of SCs $S$, written $D \models S$, if $SAT(D) \subseteq SAT(S)$.

Let $t \in T_\Sigma$ and $a, b \in \Sigma$ be element names. Tree $t$ *satisfies* the *functional constraint* (FC) $a \downarrow b$ if no node labelled $a$ in $t$ has two distinct children labelled with $b$. If $C$ is a set of SCs and FCs over $\Sigma$, then $SAT(C)$ denotes the set of trees in $T_\Sigma$ which satisfy each SC and FC in $C$. The definitions for a regular expression and a DTD implying an FC, or a set of FCs, are analogous to those for SCs.

## 3 Decidability of Containment under DTDs

When the XPath fragment includes $\text{XP}^{\{[],*,//\}}$ along with disjunction, variable binding and equality testing, Deutsch and Tannen state that the decidability of containment in the presence of DTDs is an open problem [7]. In this section, we contribute to this investigation by showing that containment of $\text{XP}^{\{[],*,//\}}$ queries under DTDs is decidable. (In fact, including disjunction in the XPath fragment as well does not change the decidability result.) We do this by showing that, given a query $Q$ in $\text{XP}^{\{[],*,//\}}$ and an alphabet $\Sigma$ for a DTD $D$, we can construct a regular tree grammar (RTG) $G$ such that the set of trees generated by the grammar is precisely the set of trees that satisfy $Q$. The result then follows from the facts that DTDs are a special case of RTGs, that RTGs are closed under intersection [6], and that containment is decidable (and EXPTIME-complete) for RTGs [4,12,13]. This same result is proved independently in [10].

The definition of regular tree grammars we use is similar to that in [6]. A *regular tree grammar* (RTG) $G$ is a 4-tuple $\langle \Sigma, N, P, n_0 \rangle$, where

1. $\Sigma$ is a finite set of element names,
2. $N$ is a finite set of nonterminals,
3. $P$ is a finite set of productions of the form $n \to a(R)$, where $n \in N$, $a \in \Sigma$, and $R$ is a regular expression over $N$,
4. $n_0 \in N$ is the start symbol.

RTG $G$ *allows* a document tree $t$ if $t$ can be derived from $n_0$ by applying productions from $P$ and $t$ does not contain any nonterminals. A production $n \to a(R)$ is applied to a tree $t$ by replacing the nonterminal $n$ in $t$ by a tree $a(t_1, \ldots, t_k)$, where $t_1 \cdots t_k \in L(R)$. The *regular tree language* $L(G)$ is the set of document trees allowed by the grammar $G$. Given two RTGs $G_1$ and $G_2$, we write $G_1 \supseteq G_2$ if $L(G_1) \supseteq L(G_2)$.

Given a query $Q$ in $\mathrm{XP}^{\{[\,],*,//\}}$, we want to derive an RTG $G$ such that $L(G) = SAT_\Sigma(Q)$, the set of trees over $\Sigma$ satisfying $Q$. We can use this to decide $D$-containment for queries $Q_1$ and $Q_2$ as described below.

Firstly, Miklau and Suciu show that there is a translation from $\mathrm{XP}^{\{[\,],*,//\}}$ queries over $\Sigma$ to Boolean $\mathrm{XP}^{\{[\,],*,//\}}$ queries over an extended alphabet $\Sigma'$ such that for any $\mathrm{XP}^{\{[\,],*,//\}}$ queries $Q_1$ and $Q_2$ and their translations $Q_1'$ and $Q_2'$, $Q_1 \supseteq Q_2$ if and only if $Q_1' \supseteq Q_2'$ [9]. Now let $G_1$ and $G_2$ be RTGs such that $L(G_1) = SAT_{\Sigma'}(Q_1')$ and $L(G_2) = SAT_{\Sigma'}(Q_2')$. Then $Q_1' \supseteq_{SAT(D)} Q_2'$ if and only if $D \cap G_1 \supseteq D \cap G_2$. So for the rest of this section, we assume that all queries are Boolean.

Let alphabet $\Sigma = \{a_1, \ldots a_k\}$. As a shorthand notation, we write

$$n \to \Sigma\ (r)$$

for the set of productions

$$\begin{aligned} n &\to a_1\ (r) \\ &\vdots \\ n &\to a_k\ (r) \end{aligned}$$

In order to generate an arbitrary tree over $\Sigma$, we define a nonterminal $n_\Sigma$ by the (shorthand) production:

$$n_\Sigma \to \Sigma\ (n_\Sigma^*)$$

*Example 4.* Given query $Q = a/b$ over alphabet $\Sigma$, the productions for the RTG corresponding to $Q$ are

$$\begin{aligned} n_a &\to a\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b &\to b\ (n_\Sigma^*) \end{aligned}$$

since $Q$ matches trees in which nodes labelled $a$ can have arbitrary children labelled with elements from $\Sigma$ that precede and/or follow the child labelled $b$, which in turn can be the root of an arbitrary subtree. □

We also have to take into account that order in RTGs is significant. So for a query which has a node with more than one child, we need to represent all possible orderings of the children in the RTG. To make this more manageable, we use the & operator from SGML as a shorthand notation: for all pairs of symbols $a$ and $b$ in $\Sigma$, $a \,\&\, b \equiv ((a,b) \mid (b,a))$. Descendant edges in queries are modelled by recursive productions in the RTG.

*Example 5.* The productions for query $a[b][c]$ are

$$\begin{aligned} n_a &\to a\ (n_\Sigma^* \,\&\, n_b \,\&\, n_\Sigma^* \,\&\, n_c \,\&\, n_\Sigma^*) \\ n_b &\to b\ (n_\Sigma^*) \\ n_c &\to c\ (n_\Sigma^*) \end{aligned}$$

while those for query $a//b$ are

$$\begin{aligned} n_a &\to a\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b &\to b\ (n_\Sigma^*) \\ n_b &\to \Sigma\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \end{aligned}$$

□

Given an alphabet $\Sigma = \{a_1, \ldots a_k, *\}$ and a query $Q$ in $\mathrm{XP}^{\{[\,],*,//\}}$ with $m$ nodes, we construct an RTG $G$ from $Q$ as follows. First number each node in $Q$ uniquely, with the root node numbered 1. The RTG $G$ *corresponding to* $Q$ is given by $\langle \Sigma, N, P, n_1 \rangle$, where $N = \{n_1, \ldots, n_m, n_\Sigma\}$, and $P$ is constructed inductively as follows.

1. If node $i$ in $Q$ is a leaf node, then $P$ includes

   $$n_i \to a_j\ (n_\Sigma^*)$$

   if $i$ has label $a_j \in \Sigma$, or

   $$n_i \to \Sigma\ (n_\Sigma^*)$$

   if $i$ has label $*$.
2. If node $i$ in $Q$ has child nodes $j_1, \ldots, j_m$, then $P$ includes

   $$n_i \to a_l\ (n_\Sigma^* \,\&\, n_{j_1} \,\&\, n_\Sigma^* \,\&\cdots\, \&\, n_\Sigma^* \,\&\, n_{j_m} \,\&\, n_\Sigma^*)$$

   if $i$ has label $a_l \in \Sigma$, or

   $$n_i \to \Sigma\ (n_\Sigma^* \,\&\, n_{j_1} \,\&\, n_\Sigma^* \,\&\cdots\, \&\, n_\Sigma^* \,\&\, n_{j_m} \,\&\, n_\Sigma^*)$$

   if $i$ has label $*$.
3. If node $i$ in $Q$ is connected to its parent by a descendant edge, then $P$ includes

   $$n_i \to \Sigma\ (n_\Sigma^*\ n_i\ n_\Sigma^*)$$

We now define the notion of a node-disjoint embedding, which is useful when defining the relationship between query $Q$ and the trees generated by the RTG corresponding to $Q$. A *node-disjoint embedding* from query $Q$ to tree $t$ is a pair $h = (h_1, h_2)$, where $h_1$ is a one-to-one mapping from the nodes of $Q$ to the nodes of $t$ such that (1) $h_1$ maps $root(Q)$ to $root(t)$, and (2) for each node $x$ in $Q$, $\lambda(x) = *$ or $\lambda(x) = \lambda(h_1(x))$, and $h_2$ is a one-to-one mapping from the edges of $Q$ to paths of $t$ such that (3) for each node $x$ and $y$ in $Q$, if $e = (x, y)$ is a child edge in $Q$, then $h_2(e) = (h_1(x), h_1(y))$ is an edge in $t$, (4) for each node $x$ and $y$ in $Q$, if $e = (x, y)$ is a descendant edge in $Q$, then $h_2(e) = (h_1(x), \ldots, h_1(y))$ is a path in $t$. We also require that all paths in the image of $h_2$ are node-disjoint, except that two paths may have the same initial node. It follows immediately from the definition that, if $h = (h_1, h_2)$ is a node-disjoint embedding from query $Q$ to tree $t$, then $h_1$ is an embedding from $Q$ to $t$.

**Lemma 1.** *Let $Q$ be a query over $\Sigma$ in $XP^{\{[],*,//\}}$, $G$ be the RTG corresponding to $Q$, and $t$ be a tree over $\Sigma$. There is a node-disjoint embedding from $Q$ to $t$ if and only if $t \in L(G)$.*

A *contraction* of a query $Q_1$ is a query $Q_2$ comprising a subset of the nodes of $Q_1$ such that there is a containment mapping from $Q_1$ to $Q_2$. Fig. 2 gives an example of some contractions for a query in $\mathrm{XP}^{\{[],*,//\}}$.

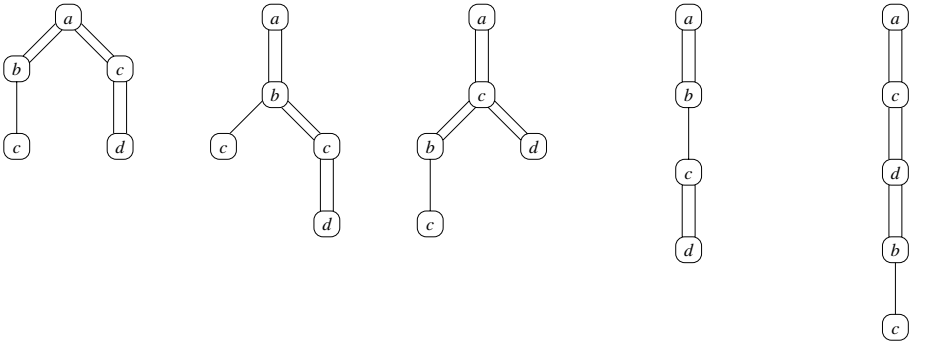



**Fig. 2.** Some contractions for the XPath query $a[.//b/c][.//c//d]$.

**Lemma 2.** *Let $Q_1$ be a query over $\Sigma$ in $XP^{\{[],*,//\}}$ and $t$ be a tree over $\Sigma$. If there is an embedding from $Q_1$ to $t$, then there is a contraction $Q_2$ of $Q_1$ such that there is a node-disjoint embedding from $Q_2$ to $t$.*

It is straightforward to construct an RTG $G$ from a set of contractions for a query $Q$. Given a set $C$ of contractions for $Q$, we number the nodes of queries uniquely throughout $C$, except that the root node for each contraction is numbered 1. The set of productions of $G$ is then simply the union of the sets of productions for each of the individual contractions in $C$.

**Lemma 3.** *Let $Q$ be a query over $\Sigma$ in $XP^{\{[],*,//\}}$ and $G$ be the RTG corresponding to the set of contractions of $Q$. Then $L(G) = SAT_\Sigma(Q)$.*

*Example 6.* Consider the following pair of queries $Q_1 = a/*//b$ and $Q_2 = a//*/b$ from [9]. Although $Q_1$ and $Q_2$ are equivalent, there is no containment mapping between them. Given alphabet $\Sigma$, the RTG productions for $Q_1$ and $Q_2$ are

$$\begin{array}{l} n_a \to a\ (n_\Sigma^*\ n_*\ n_\Sigma^*) \\ n_* \to \Sigma\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b \to \Sigma\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b \to b\ (n_\Sigma^*) \end{array} \quad \text{and} \quad \begin{array}{l} n_a \to a\ (n_\Sigma^*\ n_*\ n_\Sigma^*) \\ n_* \to \Sigma\ (n_\Sigma^*\ n_*\ n_\Sigma^*) \\ n_* \to \Sigma\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b \to b\ (n_\Sigma^*) \end{array}$$

respectively, Clearly, the above two sets of productions are equivalent. □

**Theorem 1.** *Let $D$ be a DTD over $\Sigma$, $Q_1$ and $Q_2$ be queries over $\Sigma$ in $XP^{\{[],*,//\}}$, and $G_1$ and $G_2$ be the RTGs corresponding to the sets of contractions of $Q_1$ and $Q_2$, respectively. Then $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $D \cap G_1 \supseteq D \cap G_2$.*

**Corollary 1.** *Containment for queries in $XP^{\{[],*,//\}}$ under DTDs is decidable and, in fact,* EXPTIME*-complete [13].*

*Example 7.* Consider the DTD $D$ from Example 2, along with queries $Q_1 = a[b[e][g]][d]$ and $Q_2 = a[b/e][b/g]][d]$. Note that $Q_1$ is a contraction of $Q_2$ and hence $Q_2 \supseteq Q_1$. Because $D$ dictates that if an $a$-node has a $d$-child, then it has at most one $b$-child, we have that $Q_1 \supseteq_{SAT(D)} Q_2$.

The productions for $G_2$ corresponding to the set of contractions of $Q_2$ are

$$\begin{array}{l} n_a \to a\ (n_\Sigma^*\ \&\ n_{b_1}\ \&\ n_\Sigma^*\ \&\ n_{b_2}\ \&\ n_\Sigma^*\ \&\ n_d\ \&\ n_\Sigma^*) \\ n_{b_1} \to b\ (n_\Sigma^*\ \&\ n_e\ \&\ n_\Sigma^*) \\ n_{b_2} \to b\ (n_\Sigma^*\ \&\ n_g\ \&\ n_\Sigma^*) \\ n_a \to a\ (n_\Sigma^*\ \&\ n_b\ \&\ n_\Sigma^*\ \&\ n_d\ \&\ n_\Sigma^*) \\ n_b \to b\ (n_\Sigma^*\ \&\ n_e\ \&\ n_\Sigma^*\ \&\ n_g\ \&\ n_\Sigma^*) \\ n_d \to d\ (n_\Sigma^*) \\ n_e \to e\ (n_\Sigma^*) \\ n_g \to g\ (n_\Sigma^*) \end{array}$$

while those for the RTG $G_1$, corresponding to the set of contractions of query $Q_1$, comprise the last 5 productions above.

DTD $D$ can simply be represented as the following RTG, which we also denote by $D$

$$\begin{array}{l} n_a \to a\ ((n_b\ n_b\ n_c)\ |\ (n_d\ n_b)) \\ n_b \to b\ ((n_e\ |\ n_f)\ (n_g\ |\ n_h)) \end{array}$$

along with productions of the form

$$n_x \rightarrow x\ (\epsilon)$$

for $x$ instantiated to each of $c, d, e, f, g$ and $h$. Now $D \cap G_1$ is

$$\begin{array}{ll} n_a \rightarrow a\ (n_d\ n_b) & n_d \rightarrow d\ (\epsilon) \\ n_b \rightarrow b\ (n_e\ n_g) & n_e \rightarrow e\ (\epsilon) \\ & n_g \rightarrow g\ (\epsilon) \end{array}$$

It turns out that $D \cap G_1 \equiv D \cap G_2$, since the first production for $n_a$ in $G_2$ cannot be satisfied by any tree that satisfies $D$—it requires the existence of 2 children labelled with $b$ and one with $d$. We conclude that $Q_1 \supseteq_{SAT(D)} Q_2$. □

## 4 PTIME Classes

We now turn our attention to fragments of XPath for which containment under DTDs can be tested in PTIME. We restrict ourselves to studying $\text{XP}^{\{[]\}}$ and its subclasses, since it has been shown that, even for $\text{XP}^{\{[]\}}$, deciding containment under DTDs in CONP-complete [10,16]. The approach adopted in [16], related to that in [7], is to look for classes of simple constraints implied by a DTD $D$ which are necessary and sufficient to show $D$-containment. For example, it turns out that SCs are necessary and sufficient to show $D$-containment for *duplicate-free* queries in $\text{XP}^{\{[]\}}$. Unfortunately, the CONP-hardness result relates to deciding whether a DTD implies an SC.

It is shown in [16] that when DTD $D$ is *duplicate-free*, SC implication is in PTIME. In this section, we show that SCs and FCs are necessary and sufficient to decide $D$-containment for $\text{XP}^{\{[]\}}$ queries when $D$ is duplicate-free. We also show that finding the set of FCs implied by a DTD can be done in PTIME, as can deciding containment under duplicate-free DTDs for $\text{XP}^{\{[]\}}$ queries.

**Lemma 4.** *Let $D$ be a DTD, $C$ be the set of SCs and FCs implied by $D$, and $P$ and $Q$ be $XP^{\{[]\}}$ queries. If $P \equiv_{SAT(C)} Q$, then $P \equiv_{SAT(D)} Q$.*

We can find the set of FCs implied by a DTD $D$ over $\Sigma$ in PTIME. For each regular expression $R^a$ in $D$, we construct its NFA $M^a$ and find the strongly-connected components (SCCs) of $M^a$. Then $D$ implies the FC $a \downarrow b$ if and only if $b$ labels a transition in $M^a$ which is not part of any SCC. If $|\Sigma| = n$, at most $n^2$ FCs can be implied by $D$.

We are particularly interested in when SCs and FCs are *necessary* to show $D$-containment of $\text{XP}^{\{[]\}}$ queries. For this, we need to consider duplicate-free DTDs. Recall that a DTD $D$ is *duplicate-free* if in each regular expression $R^a$ in $D$, each element name appears at most once in $R^a$.

We now consider some properties of duplicate-free DTDs. Firstly, for each regular expression $R$ in a duplicate-free DTD $D$, we can construct an NFA $M$ (with $\epsilon$-transitions) accepting $L(R)$ such that no symbol labels more than one transition in $M$. This applies even if we use the operators + and ? in $R$. As a result of no symbol being repeated in $M$, each symbol in $R$ appears either an unbounded number of times in strings in $L(R)$ or else it appears at most once in any string in $L(R)$.

**Lemma 5.** *Let $D$ be a duplicate-free DTD. If in every tree in $SAT(D)$ whenever a node with label $a$ has children with labels $b_1, \ldots, b_n$ it has a child with label $c$, then $D \models a : \{b_1, \ldots, b_n\} \Downarrow c$.*

*Example 8.* The result of Lemma 5 does not hold necessarily hold for DTDs which are not duplicate-free. Consider the DTD $D$ from Example 2. Every tree in $SAT(D)$ with a node labelled $a$ that has two children labelled $b$ also has a child labelled $c$. This means, for example, that

$$a[b/e][b/f][c] \supseteq_{SAT(D)} a[b/e][b/f]$$

because the two copies of $b$ in each of these expressions must always map to distinct nodes in any document tree (we have to consult $D$ in order to determine this). However, the only (nontrivial) SC for $a$ implied by $D$ is $a : \emptyset \Downarrow b$. □

In order to test $C$-containment, for a set $C$ of SCs and FCs, we introduce a variation of the *chase* [8], a procedure for applying a set of SCs and FCs to a query in $\mathrm{XP}^{\{[]\}}$. Let $P$ be a query in $\mathrm{XP}^{\{[]\}}$ and $C$ be the set of SCs and FCs implied by a DTD $D$.

1. Let $s \in C$ be a nontrivial SC of the form $a : B \Downarrow c$, where $B = \{b_1, \ldots, b_n\}$. Let $u$ be a node in $P$ with children $v_1, \ldots, v_n$ such that $\lambda(u) = a$, $\lambda(v_i) = b_i$, $1 \leq i \leq n$, and $u$ does not have a child labelled $c$. The *result of applying* the SC $s$ to $u$ in $P$ is a query which has the same nodes and edges as $P$ and in addition has a child of $u$ labelled $c$.
2. Let $f \in C$ be an FC of the form $a \downarrow b$. Let $u$ be a node in $P$ with distinct children $v$ and $w$ such that $\lambda(u) = a$ and $\lambda(v) = \lambda(w) = b$. The *result of applying* the FC $f$ to $u$ in $P$ is a query $\theta(P)$, where $\theta$ maps $v$ to $w$ and is the identity elsewhere.

A *chasing sequence* of $P$ by $C$ is a sequence $P = P_0, \ldots, P_k$ such that for each $0 \leq i \leq k-1$, $P_{i+1}$ is the result of applying some SC or FC in $C$ to $P_i$, and no SC or FC can be applied to $P_k$. Note that the chasing sequence is finite. The *chase* of $P$ by $C$, denoted $chase_C(P)$, is $P_k$.

For the set $C$ of SCs and FCs implied by a DTD $D$ and a query $Q$ in $\mathrm{XP}^{\{[]\}}$, $chase_C(Q)$ does not necessarily satisfy $D$. One reason is that $D$ might require that one of two child nodes be present which cannot be captured by $C$. However, we do have the following.

**Lemma 6.** *Let $D$ be a duplicate-free DTD, $Q$ be an XPath query, and $C$ be the set of SCs and FCs implied by $D$. If $Q$ is $D$-satisfiable, then $chase_C(Q)$ is isomorphic to a subtree of a tree in $SAT(D)$.*

*Example 9.* Once again, the above result does not hold for DTDs which are not duplicate-free. Consider the DTD $D$ from Example 2. Although in $D$ each $a$ can have either one or two $b$ children, if an $a$ has $d$ child, then it has at most one $b$ child. For queries $P = a[d][b[e][g]]$ and $Q = a[d][b/e][b/g]$, such a constraint implies that $P \supseteq_{SAT(D)} Q$. There is no containment mapping from $P$ to $Q = chase_C(Q)$, the problem being that $chase_C(Q)$ is not isomorphic to a subtree of a tree in $SAT(D)$. □

Let $C$ be the set of SCs and FCs implied by DTD $D$. Let $Q$ be $D$-satisfiable, and $R \subseteq SAT(D)$ be the set of trees with a subtree isomorphic to $chase_C(Q)$. We call $R$ the *$C$-satisfying set* for $Q$. Each tree in $R$ has a *core* subtree which is isomorphic to $chase_C(Q)$, and each node in the core subtree is called a *core* node. Each node which is not a core node is called a *non-core* node.

**Lemma 7.** *Let $D$ be a duplicate-free DTD and $P$ and $Q$ be queries in $XP^{\{[]\}}$. Let $Q$ be $D$-satisfiable, $C$ be the set of SCs and FCs implied by $D$, and $R$ be the $C$-satisfying set for $Q$. If $P \supseteq_{SAT(D)} Q$, then, for each node $w$ in $P$, either $w$ can be mapped to a core node in every tree in $R$ or $w$ can be mapped to a non-core node in every tree in $R$.*

**Lemma 8.** *For $XP^{\{[]\}}$ queries $P$ and $Q$, $P \supseteq_{SAT(C)} Q$ if and only if $P \supseteq chase_C(Q)$.*

**Theorem 2.** *Let $D$ be a duplicate-free DTD and $C$ be the set of SCs and FCs implied by $D$. For $XP^{\{[]\}}$ queries $P$ and $Q$, $P \supseteq_{SAT(D)} Q$ if and only if $P \supseteq_{SAT(C)} Q$.*

Let $D$ be a duplicate-free DTD and $C$ be the set of SCs and FCs implied by $D$. The number of constraints in $C$ can be exponential in the size of $D$, and $chase_C(Q)$ can have a number of nodes which is exponential in the number of constraints in $C$. However, in order to test $P \supseteq_{SAT(D)} Q$ for $\text{XP}^{\{[]\}}$ queries $P$ and $Q$, we do not have to generate all of $C$ from $D$, nor do we have to chase $Q$ with every constraint in $C$.

Instead we can first chase $Q$ with each of the FCs in $C$, a step which can clearly be done in PTIME. Since applying an SC to $Q$ never introduces a child with the same label as one of its siblings, there is no need to apply FCs after applying SCs. Now simultaneous top-down traversals of $P$ and $Q$ can determine nodes $x$ in $P$ and $u$ in $Q$ such that $\lambda(x) = \lambda(u)$, $x$ has child $y$ and $u$ has no child with label $\lambda(y)$. Let $B$ denote the set of labels of children of $u$ in $Q$. We can then determine if $D$ implies the SC $\lambda(u) : B \Downarrow \lambda(y)$ in PTIME, since $D$ is duplicate-free [16]. If so, then we add a node $v$ as a child of $u$ with $\lambda(v) = \lambda(y)$. In this way, if $P$ has $n$ nodes, we can never add more than $n^2$ nodes to $Q$.

## 5 Limitations of Constraints

In this section, we address the question of whether there are simple constraints other than sibling and functional constraints that are necessary and sufficient for deciding $D$-containment in PTIME when neither the DTD nor the queries in $\text{XP}^{\{[]\}}$ are duplicate-free.

Given a regular expression $R$, let $w \in L(R)$. Since queries in $\text{XP}^{\{[]\}}$ cannot query the order of symbols in $w$, we adopt a bag representation to indicate the insignificance of ordering in the symbols of $w$: the notation $[w]$ denotes the bag of symbols appearing in $w$. If symbol $a_i$ appears $m_i$ times in $w$, $1 \leq i \leq k$, then

$[w] = \{a_1^{m_1}, \ldots, a_k^{m_k}\}$. The *unordered regular language* denoted by $R$, written $UL(R)$, is defined as $UL(R) = \{[w] \mid w \in L(R)\}$.

Let $\Sigma$ be the set of symbols used in $R$, and $w$ be an arbitrary string over $\Sigma$. We now show how to construct a DTD $D$ that uses $R$, along with queries $Q_1$ and $Q_2$ in $\text{XP}^{\{[]\}}$, such that $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $[w] \notin UL(R)$. Of course none of $D$, $Q_1$ or $Q_2$ is duplicate-free. This effectively demonstrates that using constraints less powerful than those which characterize unordered regular languages cannot provide a necessary and sufficient condition for query containment for $\text{XP}^{\{[]\}}$.

**Theorem 3.** *Let $R$ be a regular expression, $\Sigma$ be the set of symbols used in $R$, and $w$ be a string over $\Sigma$. There is a DTD $D$ with content model $R$ for some element, along with queries $Q_1$ and $Q_2$ in $XP^{\{[]\}}$, such that $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $[w] \notin UL(R)$.*

## 6 Related Work

The most recent related work is in the papers [1,7,9,10,16]. For containment of queries in the absence of constraints, containment was shown to be in PTIME for $\text{XP}^{\{[],//\}}$ in [1], while it was shown to be CONP-complete for $\text{XP}^{\{[],*,//\}}$ in [9].

Containment under constraints was shown to be undecidable in [7], when the XPath fragment includes $\text{XP}^{\{[],*,//\}}$ along with disjunction, variable binding and equality testing, and the constraints include so-called bounded simple XPath integrity constraints (SXICs) and those (unbounded) constraints implied by DTDs. On the other hand, containment under DTDs was shown to be CONP-complete for $\text{XP}^{\{[]\}}$ in [10,16]. In [1], child constraints along with *descendant* and *type co-occurrence* constraints, are used to minimise queries in $\text{XP}^{\{[],//\}}$ in PTIME. A comprehensive classification of the complexity of containment of XPath fragments under DTDs is given in [10].

Earlier relevant work includes that by Calvanese *et al.* who proved that containment of conjunctions of regular path queries, a language more powerful than $\text{XP}^{\{[],*,//\}}$, is decidable [5]. Papakonstantinou and Vassalos studied rewriting a query expressed in a language called TSL in terms of a set of views [11]. They include some rewritings based on DTD constraints. Finally, Böhm *et al.* derived constraints from DTDs in order to optimise expressions in the PAT algebra [3].

## 7 Conclusion

In this paper, we have studied the problem of query containment under DTDs for various fragments of XPath. By showing that containment under DTDs is decidable for $\text{XP}^{\{[],*,//\}}$, we have contributed to solving the open problem stated in [7]. We have also identified that containment under duplicate-free DTDs for queries in $\text{XP}^{\{[]\}}$ is tractable, while for queries and DTDs which are not duplicate-free, it remains intractable.

Further work is obviously needed to solve the issue of decidability of containment under DTDs for larger XPath fragments.

## References


1. S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. In *Proc. ACM SIGMOD Conf.*, pages 497–508, 2001.
2. J. Bailey, A. Poulovassilis, and P. T. Wood. An event-condition-action language for XML. In *Proc. 11th Int. Conf. on the World Wide Web*, pages 486–495, 2002.
3. K. Böhm, K. Aberer, M. T. Özsu, and K. Gayer. Query optimization for structured documents based on knowledge of the document type definition. In *Proc. IEEE Advances in Digital Libraries*, pages 196–205, 1998.
4. A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets. Available at `ftp://ftp11.informatik.tu-muenchen.de/pub/misc/caterpillars/`, 1998.
5. D. Calvanese, G. de Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. 17th ACM Symp. on Principles of Databases Systems*, pages 149–158, 1998.
6. B. Chidlovskii. Using regular tree automata as XML schemas. In *Proc. IEEE Advances in Digital Libraries*, pages 89–104, 2000.
7. A. Deutsch and V. Tannen. Containment and integrity constraints for XPath fragments. In *Proc. 8th Int. Workshop on Knowledge Representation Meets Databases*, 2001.
8. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. on Database Syst.*, 4(4):455–469, 1979.
9. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21st ACM Symp. on Principles of Databases Systems*, 2002.
10. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. 9th Int. Conf. on Database Theory*, 2003.
11. Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proc. ACM SIGMOD Conf.*, pages 455–466, 1999.
12. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 19th ACM Symp. on Principles of Databases Systems*, pages 35–46, 2000.
13. H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Computing*, 19(3):424–437, June 1990.
14. P. Wadler. A formal semantics of patterns in XSLT. In *Markup Technologies*, pages 183–202, 1999.
15. P. T. Wood. On the equivalence of XML patterns. In *Proc. 1st Int. Conf. on Computational Logic*, LNAI 1861, pages 1152–1166, 2000. Springer-Verlag.
16. P. T. Wood. Minimising simple XPath expressions. In *Proc. WebDB 2001: Fourth Int. Workshop on the Web and Databases*, pages 13–18, 2001.
17. World Wide Web Consortium. XML Path Language (XPath), Version 1.0. See `http://www.w3.org/TR/xpath`, November 1999. W3C Recommendation.
18. World Wide Web Consortium. XSL Transformations (XSLT), Version 1.0. See `http://www.w3.org/TR/xslt`, November 1999. W3C Recommendation.
19. World Wide Web Consortium. XQuery 1.0: An XML Query Language. See `http://www.w3.org/TR/xquery`, June 2001. W3C Working Draft.
20. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. 7th Int. Conf. on Very Large Data Bases*, pages 82–94, 1981.

# XPath Containment in the Presence of Disjunction, DTDs, and Variables

Frank Neven[1] and Thomas Schwentick[2]

[1] University of Limburg
frank.neven@luc.ac.be
[2] Philipps-Universität Marburg
tick@mathematik.uni-marburg.de

**Abstract.** XPath is a simple language for navigating an XML tree and returning a set of answer nodes. The focus in this paper is on the complexity of the containment problem for various fragments of XPath. In addition to the basic operations (child, descendant, filter, and wildcard), we consider disjunction, DTDs and variables. W.r.t. variables we study two semantics: (1) the value of variables is given by an outer context; (2) the value of variables is defined existentially. We establish an almost complete classification of the complexity of the containment problem w.r.t. these fragments.

## 1 Introduction

XPath is a simple language for navigating an XML document and selecting a set of element nodes [5]. Actually, XPath is the main XML selection language. Indeed, XPath expressions are used, for instance, as basic patterns in several XML query languages like XQuery [3] and XSLT [2,6]; they are used in XML Schema to define keys [7], and in XLink [9] and XPointer [8] to reference elements in external documents. In every such context an instance of the containment problem is present: optimizing XPath expressions can be accomplished by an algorithm for containment, and XSLT rule selection and inference of keys based on XPath expressions again reduces to containment. In this paper we focus on the complexity of the containment problem of various fragments of XPath. In particular, we consider disjunction, DTDs and variables. The complexity of XPath evaluation has recently been studied in [11,12]. The XPath containment problem already attracted quite some attention [1,10,14,15,23,21]. We next discuss the known results together with our own contributions. The initial XPath fragment consists of the following operators: /, //, [], $*$, | which denote child, descendant, filter, wildcard, and disjunction, respectively. We indicate the fragment we use by listing the allowed operators. For instance, XP(/,//) denotes the XPath fragment where only child and descendant are allowed.

Among other results, Miklau and Suciu obtained that containment of XP(/,//, [],$*$) is coNP-complete [14]. For this result it does not matter whether one considers XML documents over a finite or an infinite alphabet. We show

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 315–329, 2003.

that adding disjunction to this fragment does not make the containment problem harder when the input alphabet is infinite. The proof is an extension of their canonical model technique. However, when XML documents are restricted to a finite alphabet the containment problem turns PSPACE-complete. The reason for this complexity jump is that when the alphabet is finite, disjunction allows to express negation (like is the case with regular expressions), and negation allows for a reduction from CORRIDOR TILING [4]. The upper bound is obtained by a reduction to the containment problem of alternating tree automata on bounded trees.

Deutsch and Tannen consider XPath containment in the presence of DTDs and Simple XPath Integrity Constraints (SXICs) [10]. They obtain that this problem is undecidable in general and in the presence of bounded SXICs and DTDs. When only DTDs are present they have a PSPACE lower bound and leave the exact complexity as an open question. We show that containment testing for XP(DTD,/,//,[],∗,|) is in EXPTIME and obtain that containment of XP(DTD,/,//,|) and XP(DTD,/,//,[],∗) is hard for EXPTIME. The upper bound is obtained by a reduction to containment of unranked tree automata [19]. The presence of the DTD allows for a reduction from TWO-PLAYER CORRIDOR TILING [4]. We do not know much about the complexity of more restrictive fragments in the presence of DTDs. In fact, we can only prove that containment of XP(DTD,/,[]) is CONP-complete and containment of XP(DTD,//,[]) is CONP-hard. It is not clear whether or how the upper bound proof can be extended to include, for instance, the descendant operator. Further, we show that in the presence of very simple DTDs and node-set inequality the containment problem is undecidable. The DTD can be eliminated when a modest form of negation is allowed: negation that can express that a node cannot have a certain label.

The XPath recommendation allows variables to be used in XPath expressions on which equality tests can be performed. For instance, $//a[\$x = @b][\$y \neq @c]$ selects all $a$-descendants whose $b$-attribute equals the value of variable $\$x$ and whose $c$-attribute differs from the value of variable $\$y$. However, under the XPath semantics the value of all variables should be specified by the outer context (e.g., in the XSLT template in which the pattern is issued). So the semantics of a pattern is defined w.r.t. a variable mapping. We show that the complexity of containment is PSPACE-complete under this semantics. For the lower bound, it suffices to observe that with variables a finite alphabet can be simulated. We obtain the upper bound by reducing the containment problem to the containment of several patterns without variables.

In addition to the XPath semantics, Deutsch and Tannen [10] considered an existential semantics for variables: a pattern matches a document if there is an assignment for the variables such that the pattern matches w.r.t. this assignment. W.r.t. the existential semantics they showed that containment of XP(/,//,[],∗,vars) and XP(/,//,[],|,vars) is $\Pi_2^P$-hard, and that containment of XP(/,//, [],|,vars) under fixed bounded SXICs is in $\Pi_2^P$. We extend their result by showing that containment of XP(/,//,[],|,vars,≠), that is, inequality tests on

variables and attribute values are allowed, remains in $\Pi_2^P$. Interestingly, when $*$ is added, the problem turns undecidable.

In a recent paper, Wood obtained that containment of XP(/,//,[],*) in the presence of DTDs is decidable [21]. He also studies conditions for which containment under DTDs is in PTIME. Benedikt, Fan, and Kuper study the expressive power and closure properties of fragments of XPath [1]. They also consider sound and complete axiom systems and normal forms for some of these fragments.

This paper is organized as follows. In Section 2, we define DTDs and the basic XPath fragments. In Section 3, 4, and 5 we consider disjunction, DTDs, and variables, respectively. We conclude in Section 6.

Due to space limitations we only provide sketches of proofs.

## 2 Preliminaries

In the present section, we define trees, DTDs, and the core of XPath. For the rest of this paper we fix a recursively enumerable infinite alphabet $\Sigma$ and a recursively enumerable infinite set of data values $\mathbf{D}$. $A$ is always a finite set of attributes. An XML document is faithfully modelled by a finite $\Sigma$-tree where the attributes of the nodes have $\mathbf{D}$-values.

Formally, a *tree domain* $\tau$ *over* $\mathbb{N}$ is a subset of $\mathbb{N}^*$, such that if $v \cdot i \in \tau$, where $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in \tau$. Here, $\mathbb{N}$ denotes the set of natural numbers without zero. If $i > 1$ then also $v \cdot (i-1) \in \tau$. The empty sequence, denoted by $\varepsilon$, represents the root. We call the elements of $\tau$ *vertices*. A vertex $w$ is a *child* of a vertex $v$ (and $v$ the *parent* of $w$) if $vi = w$, for some $i$.

**Definition 1.** A $(\Sigma, A)$-*tree* is a triple $t = (\mathrm{dom}(t), \mathrm{lab}_t, \lambda_t)$, where $\mathrm{dom}(t)$ is a tree domain over $\mathbb{N}$, $\mathrm{lab}_t : \mathrm{dom}(t) \to \Sigma$, and, for each $a \in A$, $\lambda_t^a : \mathrm{dom}(t) \to \mathbf{D}$ are functions. Intuitively, $\mathrm{lab}_t(v)$ is the label of $v$, while $\lambda_t^a(v)$ is the value of $v$'s $a$-attribute.

When $\Sigma$ and $A$ are clear from the context or not important, we sometimes say tree rather than $(\Sigma, A)$-tree. Of course, in real XML documents not every element has the same attributes. Further, there can be nodes with mixed content, but these can easily be modeled by using dummy intermediate nodes [2]. We only make use of attributes in Section 5.

We formalize a DTD as a context-free grammar with regular expressions on the right-hand side of rules.

**Definition 2.** A *DTD* is a tuple $(d, S_d, \Sigma_d)$ where $\Sigma_d$ is a finite subset of $\Sigma$, $S_d \in \Sigma_d$ is the start symbol, and $d$ is a mapping from $\Sigma_d$ to the set of regular expressions over $\Sigma_d$. A tree $t$ *matches* a DTD $d$ iff $\mathrm{lab}_t(\varepsilon) = S_d$ and for every $u \in \mathrm{dom}(t)$ with $n$ children, $\mathrm{lab}_t(u1) \cdots \mathrm{lab}_t(un) \in L(d(\mathrm{lab}(u)))$. We denote by $L(d)$ the set of all trees that satisfy $d$.

Note that DTDs do not constrain the value of attributes in any way. We usually refer to a DTD by $d$ rather than $(d, S_d, \Sigma_d)$.

We next define the core fragment of XPath that we will consider in Sections 3 and 4.

**Definition 3.** An XP-*expression* is an expression defined by the following grammar:

$$\begin{array}{rl} p := p_1 \mid p_2 & \text{(disjunction)} \\ \mid /p & \text{(root)} \\ \mid //p & \text{(descendant)} \\ \mid p_1/p_2 & \text{(child)} \\ \mid p_1//p_2 & \text{(descendant)} \\ \mid p_1[p_2] & \text{(filter)} \\ \mid \sigma & \text{(element test)} \\ \mid * & \text{(wildcard)} \end{array}$$

Here, $\sigma \in \Sigma$.

It remains to define the semantics of XP expressions. In brief, every XP expression $p$ induces a mapping $[\![p]\!]_t : \mathrm{dom}(t) \mapsto 2^{\mathrm{dom}(t)}$. This mapping is defined w.r.t. a tree $t$. We inductively define $[\![p]\!]_t$ as follows: for all $u \in \mathrm{dom}(t)$,

- $[\![p_1 \mid p_2]\!]_t(u) := [\![p_1]\!]_t(u) \cup [\![p_2]\!]_t(u)$;
- $[\![/p]\!]_t(u) := \begin{cases} [\![p]\!]_t(\varepsilon) \text{ if } u = \varepsilon; \\ \emptyset \qquad \text{otherwise;} \end{cases}$
- $[\![//p]\!]_t(u) := \{v \mid \exists w : v \in [\![p]\!]_t(w)\}$;
- $[\![p_1/p_2]\!]_t(u) := \{v \mid \exists w \in \mathbb{N}^*, z \in \mathbb{N} : w \in [\![p_1]\!]_t(u) \text{ and } v \in [\![p_2]\!]_t(wz)\}$;
- $[\![p_1//p_2]\!]_t(u) := \{v \mid \exists w \in \mathbb{N}^*, z \in \mathbb{N}^* \setminus \{\varepsilon\} : w \in [\![p_1]\!]_t(u) \text{ and } v \in [\![p_2]\!]_t(wz)\}$;
- $[\![p_1[p_2]]\!]_t(u) := \{v \mid v \in [\![p_1]\!]_t(u) \text{ and } [\![p_2]\!]_t(v) \neq \emptyset\}$;
- $[\![*]\!]_t(u) := \{u\}$;
- $[\![\sigma]\!]_t(u) := \begin{cases} \{u\} \text{ if } \mathrm{lab}_t(u) = \sigma; \\ \emptyset \quad \text{otherwise;} \end{cases}$

Obviously, XPath expressions express unary queries. However, we can also use expressions for Boolean queries by testing whether $[\![p]\!]_t(\varepsilon) \neq \emptyset$. We denote the latter also by $t \models p$ or say that $t$ matches $p$. Like in [14], we can reduce containment testing of unary queries to containment of Boolean queries by introducing new labels.

**Definition 4.** We say that $p$ is *contained* in $q$, denoted $p \subseteq q$, if for all $t$, $t \models p$ implies $t \models q$. For a DTD $d$, $p$ is *contained* in $q$ *w.r.t.* $d$, denoted $p \subseteq_d q$, if for all $t \in L(d)$, $t \models p$ implies $t \models q$.

As already mentioned in the introduction, we denote the fragment of XP under consideration by listing the allowed operators. Element test is always allowed.

In Definition 3, we allow absolute expressions, as opposed to relative ones, to appear within filter expressions. For instance, in the expression $//a[/c//b]$, $/c//b$ looks for a $b$ starting at the children of the $c$-labeled root rather than starting from $a$. When no disjunction is present we can always move such expressions to the top and introduce a new root symbol. For instance, suppose we are given $//a[/c//b]$ and $//a$ then we take the expressions $/\#[//a][/c//b]$ and $/\#//a$. In the proofs for fragments where disjunction is present, we always go through their

DNF, that is, for every expression $q$ we take the equivalent expression $q_1 \mid \cdots \mid q_n$ where no $q_i$ contains disjunction. In each of the disjuncts we can move absolute expressions to the root. For this reason, we do not deal with absolute filter expression in the upper bound proofs of this paper. But it should be pointed out that we make significant use of absolute path expressions in the proof of undecidability in the presence of DTDs and node-set inequality (Theorem 10).

In some proofs, we view patterns $p$ from XP(/,//,[],$*$) as *tree patterns* as described by Miklau and Suciu [14]. From this point of view a tree $t$ matches a pattern $p$ iff there is a homomorphism from (the tree pattern associated with) $p$ to $t$, i.e., a mapping which respects labels, child and descendant, (and does not care about $*$). For example, the pattern $a/b//c[d][*/e]$ corresponds to the tree pattern in Figure 1. Single edge and double edge correspond to child and descendant relation. All nodes of the input tree that can be mapped onto the $x$-labelled node are selected.

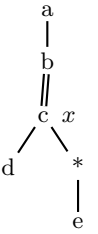



**Fig. 1.** The tree pattern corresponding to $a/b//c[d][*/e]$

## 3 The Base Cases: /, //, [], |, and $*$

Milkau and Suciu showed that XP(/,//,[],$*$) is CONP-complete [14]. In this section, we add | and show that containment remains in CONP. Containment is even hard for CONP when | is considered in isolation with / or //. In fact, during the write up of this paper, we noted that Miklau and Suciu already mention these results in their discussion section but do not provide proofs. For this reason we decided to include the proofs in this paper. Moreover, we want to stress that the reason there is an CONP upper bound is because the alphabet $\Sigma$ is infinite. Indeed, when we restrict to a finite alphabet, then XP(/,//,|) becomes hard for PSPACE.

**Theorem 5.** *1. Containment of* XP(/, //, [], $*$, |) *expressions is in* CONP*;*
*2. Containment of* XP(/, |)*-expressions is* CONP*-hard.*
*3. Containment of* XP(//, |)*-expressions is* CONP*-hard.*

*Proof.* (sketch) The hardness proofs are rather straightforward and are omitted. We proceed with the proof of (1). We develop a criterion which allows to check

in NP whether, for given patterns $p$ and $q$, $p \not\subseteq q$. Let $p$ and $q$ be fixed and let $p_1|...|p_l$ and $q_1|\dots|q_{l'}$ be the disjunctive normal forms (DNFs) of $p$ and $q$, respectively. Hence, each $p_i$ and $q_j$ is a pattern from $\mathrm{XP}(/, //, [\,], *)$. Let $n$ and $m$ denote the maximum number of nodes in a pattern $p_i$ and $q_j$, respectively. Let $T(p, q)$ be the set of trees with at most $2n(m + 2)$ nodes that are labelled with labels that occur in $p$ and with the new label # not occurring in $p$ nor in $q$. It is possible to prove the following claim:

*Claim.* $p \not\subseteq q \Leftrightarrow$ there is a $t \in T(p, q)$ such that $t \models p$ but $t \not\models q$.

It remains to show how the above criterion can be used for an NP-algorithm that checks whether $p \not\subseteq q$. The algorithm simply guesses a pattern $p_i$ from the DNF of $p$ (by nondeterministically choosing one alternative for each $|$ in $p$) and a $t \in T(n, m)$. Then it checks that $t \models p_i$ and $t \not\models q$. The latter can be done in polynomial time as shown in [12]. □

When the alphabet is finite, the containment problem becomes PSPACE-complete. Actually, the finite alphabet allows us to express that an element name in the XML document does not occur in a certain set. This is the only property we need. Therefore, if we extend the formalism with an operator $*_{\notin S}$ for a finite set $S$, expressing that any symbol but one from $S$ is allowed, then containment would also be hard for PSPACE.

**Theorem 6.** *When $\Sigma$ is finite,*

1. *containment of* $\mathrm{XP}(/, //, [\,], *, |)$*-expressions is in* PSPACE*; and*
2. *containment of* $\mathrm{XP}(/, //, |)$*-expressions is* PSPACE*-hard.*

*Proof.* (sketch) *Upper bound.* Let $k$ be a natural number. We say that a tree is *$k$-bounded* if it has at most $k$ non-unary nodes (that is, nodes with more than one child) and every node has rank at most $k$ (that is, at most $k$ children). Let $p$ and $q$ be two patterns in $\mathrm{XP}(/, //, [\,], *, |)$. Let $f(p)$ denote the number of filter expressions in $p$. It is easy to see that $p$ is contained in $q$ iff $p$ is contained in $q$ on the class of $f(p)$-bounded trees. Indeed, suppose there is a $t$ such that $t \models p$ and $t \not\models q$. Let the DNF of $p$ and $q$ be $p_1|\cdots|p_n$ and $q_1|\cdots|q_m$. Note that each disjunct has at most as many of filter expressions as the original XP expression. Let $i$ be such that $t \models p_i$. Hence, there is a $(/, //, [\,], *)$-homomorphism $h$ from $p_i$ to $t$ (as defined in the proof of Theorem 5). Let $V$ be the set of nodes on the image of $h$. Let $s$ be the tree obtained from $t$ by deleting all nodes that are not in $V$ and are not ancestors of nodes in $V$. Then, clearly, $s$ is $f(p)$-bounded, $s \models p_i$ and $s \not\models q_j$ for all $j$ (otherwise, $t \models q_j$).

By ATA we denote the class of ranked alternating top-down automata [20]. We say that an automaton is *bounded* iff there is a $k$ such that whenever a tree is accepted by the automaton it is $k$-bounded.

Given two patterns $p$ and $q$ in $\mathrm{XP}(/, //, [\,], *, |)$, let $n = f(p)$. The remainder of the proof consists of two steps: (1) we show that $p$ can be transformed into an $n$-bounded automaton $M_p$ such that $p$ and $M_p$ are equivalent on $n$-bounded

trees; (2) we show that containment of $n$-bounded automata is in PSPACE. Details are omitted.

*Lower bound.* We make use of a reduction from CORRIDOR TILING which is known to be hard for PSPACE [4]. Let $T = (D, H, V, \bar{b}, \bar{t}, n)$ be a tiling system. Here, $D = \{a_1, \ldots, a_k\}$ is a finite set of tiles; $H, V \subseteq D \times D$ are horizontal and vertical constraints, respectively; $\bar{b} = (b_1, \ldots, b_n), \bar{t} = (t_1, \ldots, t_n)$ are $n$-tuples of tiles; $n$ is a natural number in unary notation. A player places tiles on an $n \times \mathbb{N}$ board ($n$ columns, unlimited rows). On this board the bottom row is tiled with $\bar{b}$. Every placed tile should satisfy the horizontal and vertical constraints. The top row should be tiled with $\bar{t}$. The problem is to determine whether a tiling exists.

We use a string representation of the $n \times \mathbb{N}$ board where every row is delimited by # and the last symbol is \$. The XP pattern $q$ selects all strings that do not encode a tiling. As $\Sigma$ we take $D \cup \{\#, \$\}$. By $D^i$ we denote the pattern $(a_1|\cdots|a_k)$, repeated $i$ times which describes $i$ successive symbols of $D$. The pattern $p$ is //\$ assuring that the string contains the symbol \$. The pattern $q$ is the disjunction of the following patterns:

- a row has the wrong format: $\bigcup_{i=0}^{n-1} //\#D^i\#\cup\bigcup_{i=0}^{n-1} /D^i/\#\cup\bigcup_{i=0}^{n-1} //\#D^i/\$\cup //D^{n+1}$;
- \$ occurs inside the string: $//\$/(D \cup \{\$\} \cup \{\#\})$;
- the string does not begin with $b$: $\bigcup_{i=1}^{n} /b_1/\cdots/b_{i-1}/(\bigcup_{a_j \neq b_i} a_j)$;
- the string does not end with $t$: $\bigcup_{i=1}^{n}(\bigcup_{a_j \neq t_i} a_j)/t_{i+1}/\cdots/t_n/\$$
- some horizontal constraint is violated:$\bigcup_{(d_1,d_2)\notin H} //d_1/(D \cup \{\#\})^{n+1}/d_2$;
- some vertical constraint is violated: $\bigcup_{(d_1,d_2)\notin V} //d_1/d_2$.

Now, $T$ has a solution iff $p \not\subseteq q$. Clearly, if $T$ has a solution then we can take the string encoding of the tiling as a counter example for the containment of $p$ and $q$. Conversely, if $p \not\subseteq q$ then there is a, not necessarily unary, tree $t$ with one branch $s$ ending on a \$ such that $s \models p$ and $s \not\models q$. So, this branch encodes a solution for $T$. □

## 4 Containment in the Presence of DTDs

Deutsch and Tannen obtained a PSPACE lower bound on the complexity of containment in the presence of DTDs. In the general case, we prove that the complexity is EXPTIME-complete. We also have a modest NP-completeness result on the fragment using only / and []. We do not know how to extend the upper bound proof to include // or $*$. Finally, we show that adding nodeset comparisons w.r.t. $=$ and $<$ leads to undecidability. In fact, when a modest form of negation is introduced expressing that certain labels cannot appear as a child of a node, the DTD can be dispensed with. The results of this section are summarized in Table 1.

We start with a fragment in P. The lower bounds in Theorem 5 and in Theorem 8 show that this is the largest fragment whose complexity of containment w.r.t. DTDs is in P.

**Table 1.** The complexity of containment in the presence of DTDs.

| DTD | / | // | [] | \| | ∗ | complexity |
|---|---|---|---|---|---|---|
| + | + | + | | | + | in P |
| + | + | | + | | | coNP-complete |
| + | | + | + | | | coNP-hard |
| + | + | + | + | + | + | EXPTIME-complete |
| + | + | + | | + | | EXPTIME-complete |
| + | + | + | + | | + | EXPTIME-complete |
| + | + | + | + | | + | undecidable with nodeset comparisons |

**Theorem 7.** *Containment of* $\mathrm{XP}(DTD, /, //, *)$*-expressions is in* P*.*

*Proof.* (sketch) Let $d$ be a DTD and $p, q$ be patterns of $\mathrm{XP}(DTD, /, //, *)$. Note that although $p$ and $q$ only can match paths it is not sufficient to reason about single paths in trees. It might be the case that whenever a tree $t$ has a path which matches $p$, the DTD forces the tree to have a different path which matches $q$.

We construct a non-deterministic top-down tree automaton $A$ that accepts a tree $t$ if and only if (1) $t$ conforms to $d$, (2) $t \models p$, and (3) $t \not\models q$. Once $A$ is constructed it only remains to check that $A$ does not accept any tree to conclude that $p \subseteq_d q$. This can be tested in polynomial time in $|A|$. Further, the construction time and the size of $|A|$ are polynomial in the overall size of $d$, $p$, and $q$. Hence, the algorithm is indeed polynomial. Further details are omitted. □

Next, we consider a fragment in coNP. It is open whether $\mathrm{XP}(DTD, /, [])$ is a maximal fragment whose complexity of containment w.r.t. DTDs is in coNP.

**Theorem 8.** *1. Containment testing for* $\mathrm{XP}(DTD, /, [])$ *is in* coNP*.*
*2. Containment testing for* $\mathrm{XP}(DTD, /, [])$ *is* coNP*-hard.*
*3. Containment testing for* $\mathrm{XP}(DTD, //, [])$ *is* coNP*-hard.*

*Proof.* (sketch) We only prove (1). We present a nondeterministic algorithm `CheckPnotq` which checks, given a DTD $d$, a non-terminal $s$ of $d$, an $\mathrm{XP}(/, [])$-pattern $q$ and a set $P = \{p_1, \ldots, p_n\}$ of $\mathrm{XP}(/, [])$-patterns, whether there is a tree $t$ with root symbol $s$ which conforms to $d$, matches the patterns in $P$ but does not match $q$. Clearly, invoking this algorithm with $d$, $q$, $P = \{p\}$ and $s = \mathrm{root}(d)$ checks whether $p \not\subseteq_d q$.

A complication arises from the fact that the smallest counter example tree $t$ might be of exponential size due to the constraints from $d$. Hence, we can not simply guess such a counter example.

We make use of two algorithms with slightly simpler tasks. Algorithm `CheckP` checks on input $d, s, P$ whether there is a tree $t$ with root $s$ conforming to $d$ which contains all the patterns from $P$. Algorithm `Checknotq` checks on input $d, q$ whether there is a tree conforming to $d$ with a root labelled by the root symbol of $q$ which does *not* match $q$. The construction of the two algorithms is

omitted. Both work non-deterministically in polynomial time. In both algorithms and below, we make use of the following notation. For a DTD $d$ let $U(d)$ be the set of non-terminals $a$ of $d$ that are useful in the sense that there is a tree $t$ with root label $a$ that conforms to $d$. $U(d)$ can be computed in polynomial (even linear) time from $d$ by using standard methods.

Algorithm `CheckPnotq` proceeds as follows. Let $d, s, P = \{p_1, \ldots, p_n\}, q$ be an input.

- First, it checks whether all patterns in $P$ have the root symbol $s$. If this is not the case it returns FALSE.
- Next, it checks whether $q$ has the root symbol $s$. If this is not the case it calls `CheckP` with parameters $d, s, P = \{p_1, \ldots, p_n\}$ and returns TRUE iff `CheckP` does.
- It guesses a string $u$ of length at most $(|d| + 1)(l + 1)$ and verifies that $u$ conforms to the regular expression of $s$ in $d$ and that all non-terminals in $u$ are in the set $U(d)$ of useful symbols.
- It guesses a child of the root of $q$. Let $q'$ be the pattern rooted at this child.
- For each $i \in \{1, \ldots, n\}$, it guesses a mapping $f_i$ from the children $v_1, \ldots, v_m$ of the root of $p_i$ to the positions of $u$.
- For each position $j$ of $u$, which is in the image of at least one of the mappings $f_i$, it does the following
  - Let $P'$ be the vertices that are mapped to $j$.
  - If $u_j$ is the symbol of the root of $q'$ then call `CheckPnotq` recursively with parameters $d, u_j, P', q'$.
  - Otherwise call `CheckP` with parameters $d, u_j, P'$.
  - Let $s'$ be the label at the root of $q'$. If $s'$ does not occur in $P'$ but in $u$ it calls `Checknotq` with parameters $d, q'$.
- It returns TRUE iff all the subcomputations return TRUE.

Clearly, this algorithm checks nondeterministically in polynomial time whether there is a counter example conforming to $d$ which matches all patterns in $P$ but not $q$. The reasoning for the correctness is similar to the case of `CheckP`.

We note that in the above theorem, (2) and (3) were also obtained by Wood [22].

When disjunction, or filter and wildcard come in to play, the complexity raises from P and CONP to EXPTIME.

**Theorem 9.** *1. Containment testing for* $\mathrm{XP}(DTD, /, //, [\,], *, |)$ *is in* EXPTIME.
*2. Containment testing for* $\mathrm{XP}(DTD, /, //, |)$ *is hard for* EXPTIME.
*3. Containment testing for* $\mathrm{XP}(DTD, /, //, [\,], *)$ *is hard for* EXPTIME.

*Proof.* (sketch) The upper bound is shown by a translation to emptiness of an unranked tree automaton whose size is exponential in the input. See [16,17,18] for an overview of unranked tree automata.

First of all, we indicate that, for each $\mathrm{XP}(/, //, [\,], *)$-pattern $p$, one can construct in exponential time an exponential size deterministic tree automaton $A_p$ such that $A_p$ accepts a tree if and only if it matches $p$. Let $t_p$ be the tree pattern

for the expression $p$. The states of $A_p$ are pairs $(S_1, S_2)$ of sets of nodes of $t_p$. The intended meaning is as follows. If $v \in S_1$ then the subtree of the input tree rooted at the current node matches the subpattern of $t_p$ rooted at $v$. If $v \in S_2$ then there is a node below the current node of the input tree which matches the subpattern rooted at $v$. $S_2$ is used to handle descendant edges. Note that there are two sources for the exponential size of $A_p$. First of all, there is possibly an exponential number of states. Second, the regular expressions (or finite automata) that describe the transitions of $A_p$ from the children of a node to the node itself might be of exponential size. E.g., if a vertex in $t_p$ has $k$ children then the associated regular expression might be of size about $k!$.

Let now $d$ be a DTD and let $p$ and $q$ be XP$(/, //, [\,], *, |)$-patterns. Let $p = p_1 \mid \cdots \mid p_m$ and $q = q_1 \mid \cdots \mid q_l$ be disjunctive normal forms. Note that $m$ and $l$ might be exponential in $|p|$ and $|q|$, respectively (but not more). Let $A_d$ be a nondeterministic top-down automaton checking conformance with $d$. Let $A$ be the product automaton of $A_d$, the automata $A_{p_i}$ and the automata $A_{q_j}$ such that $A$ accepts, if $A_d$ accepts, at least one of the $A_{p_i}$ accepts and all the $A_{q_j}$ reject. The latter is possible as the $A_{q_j}$ are all deterministic. Clearly, $A$ is of exponential size.

Now $p \subseteq q$ if and only if $A$ does not accept any tree. This concludes the proof of (1)

The proofs of the lower bound make use of a reduction from TWO-PLAYER CORRIDOR TILING [4]. The latter problem is the extension of CORRIDOR TILING, used in the proof of Theorem 6 (1), to two players (`I` and `II`). Again the game is played on an $n \times \mathbb{N}$ board. Each player places tiles in turn. While player `I` tries to construct a corridor tiling, player `II` tries to prevent it. It is known that it is EXPTIME-complete to determine whether player `I` has a winning strategy no matter how player `II` plays. Given such a tiling system, we construct a DTD $d$ and two patterns $p$ and $q$ such that $p \not\subseteq_d q$ iff player `I` has a winning strategy. Intuitively, the DTD defines the set of all strategy trees, $p$ selects every tree, and $q$ checks whether a possible strategy tree contains an error. Details are omitted. □

The core fragment XP of XPath, defined in the previous section, leaves out many features of XPath: node-set equality, location paths, the many functions in the function library, among others. When operators from the function library like arithmetical operators or string concatenation are allowed, Moerkotte already showed that containment is undecidable [15].

It is an interesting open question to pinpoint exactly the minimal XPath fragments that have an undecidable containment problem. In the present section we show that containment already becomes undecidable in the presence of very simple DTDs when we allow node-set equality and inequality with the additional $<$ operator. In addition, we show that we can get rid of the simple DTDs when a certain kind of negation, already present in full XPath, over child labels is allowed.

We define XP$^{\text{ns}}$ as XP extended with the following rules: if $p$, $q_1$ and $q_2$ are XP$^{\text{ns}}$ expressions then $p[q_1 = q_2]$, $p[q_1 < q_2]$, $p[not(q_1 = q_2)]$, and $p[not(q_1 < q_2)]$

are $\mathrm{XP}^{\mathrm{ns}}$ expressions. To define their semantics, we introduce some notation. For a tree $t$ and a node $v \in \mathrm{dom}(t)$, define $\mathrm{yield}(t_v)$ as the string obtained by concatenating from left to right the labels of the leave nodes that are descendants of $v$. Note that this definition is in conformance with the definition of the string-value of an element node in the XPath data model [5]. For instance, if $t$ is the tree $a(b, a(c), d)$ then $\mathrm{yield}(t_\varepsilon)$ is $bcd$. We assume an ordering $<$ on $\Sigma$. The semantics is defined as follows, for $* \in \{=, <\}$,

$$[\![p[q_1 * q_2]]\!]_t(u) := \{v \mid v \in [\![p]\!]_t(u) \text{ and } \exists v_1 \in [\![q_1]\!]_t(v), \exists v_2 \in [\![q_2]\!]_t(v) \text{ such that } \mathrm{yield}(t_{v_1}) \ * \ \mathrm{yield}(t_{v_2})\};$$

$$[\![p[not(q_1 * q_2)]]\!]_t(u) := \{v \mid v \in [\![p]\!]_t(u) \text{ and } \forall v_1 \in [\![q_1]\!]_t(v), \forall v_2 \in [\![q_2]\!]_t(v) \text{ such that } \neg(\mathrm{yield}(t_{v_1}) \ * \ \mathrm{yield}(t_{v_2}))\}.$$

A *simple* DTD is a DTD where every rule is of the form $a \to b$ or $a \to c(b_1 + \cdots + b_n)$. The next proof is an involved reduction from PCP, we only provide a rough sketch.

**Theorem 10.** *Containment of $XP^{ns}$ expressions w.r.t. simple DTDs is undecidable.*

*Proof.* (sketch) We use a reduction from Post's Correspondence Problem (PCP) which is well-known to be undecidable [13]. An *instance* of PCP is a sequence of pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i, y_i \in \{a, b\}^*$ for $i = 1, \ldots, n$. This instance has a *solution* if there exist $m \in \mathbb{N}$ and $\alpha_1, \ldots, \alpha_m \in \{1, \ldots, n\}$ such that $x_{\alpha_1} \cdots x_{\alpha_m} = y_{\alpha_1} \cdots y_{\alpha_m}$.

We construct a DTD $d$, and two XPath expressions $p_1$ and $p_2$ such that $p_1 \subseteq_d p_2$ iff the PCP instance has a solution.

We consider XML trees that are almost unary trees or, equivalently, simply strings. They are of the form $u\$v$, where $\$$ is a delimiter and $u$, $v$ are strings representing a candidate solution $(x_{\alpha_1}, \ldots, x_{\alpha_m}; y_{\beta_1}, \ldots, y_{\beta_m})$ for the PCP instance in a suitable way. To check whether such a candidate is indeed a solution, we roughly have to check whether

1. $\alpha_i = \beta_i$ for each $i$, that is, corresponding pairs are taken; and
2. both strings are the same, that is, corresponding positions in $x_{\alpha_1} \cdots x_{\alpha_m}$ and $y_{\alpha_1} \cdots y_{\alpha_m}$ carry the same symbol.

To check the correspondences mentioned in (1) and (2), we make use of a double indexing system based on string-values of children nodes of the nodes of $u$ and $v$ (and therefore the trees are not literally unary). Details are omitted. □

To get rid of simple DTDs, we allow a modest form of negation. Let $\mathrm{XP}^{ns}(not)$ be the extension of $\mathrm{XP}^{ns}$ extended with the rules $p[not(a)]$ and $p[not(c) \mid not(b_1 \mid \cdots \mid b_n)]$ where $a, c, b_1, \ldots, b_n \in \Sigma$, expressing that there is a node selected by $p$ that does not have an $a$ child, and does not have a $c$-labeled child or does not have one with a label coming from $b_1, \ldots, b_n$, respectively.

**Lemma 11.** *Let $d$ be a simple DTD, let $p$ and $q$ be two $XP^{ns}$ expressions, then there is an XP(/,//,[],not) expression $q_d$ such that $p \subseteq_d q$ iff $p \subseteq q \mid q_d$.*

*Proof.* For every rule $a \to b$, let $q_a$ be the expression $//a[not(b)]$. For every rule $a \to c(b_1 + \cdots + b_n)$, let $q_a$ be the expression $//a[not(c) \mid not(b_1 \mid \cdots \mid b_n)]$. Define $q_d$ as the union of all such expressions. □

**Corollary 12.** *Containment of $XP^{ns}(not)$ expressions is undecidable.*

## 5 Containment in the Presence of Data Values

In the present section, we add attribute comparisons to our XPath fragment. Formally, we add the following rules to Definition 3:

$$\mid p[\$x = @a] \text{ (variables)}$$
$$\mid p[\$x \neq @a] \text{ (inequalities)}$$

Here, $a$ is an attribute, and $\$x, \$y$ are variables. The presence of the former rule is indicated by 'vars' the presence of the latter by '$\neq$'.

We consider two semantics. The first one corresponds to the XPath semantics and we refer to it in that way. The variable binding is defined in an outer context, not by matching the pattern with the tree. In particular, the value of a pattern is defined w.r.t. a variable assignment $\rho : X \to \mathbf{D}$ where $X$ is the set of all variables. Formally,

$$[\![p[\$x = @a]]\!]^\rho_t(u) := \{v \mid v \in [\![p]\!]^\rho_t(u) \text{ and } \rho(x) = \lambda^a_t(v)\}; \text{ and}$$

$$[\![p[\$x \neq @a]]\!]^\rho_t(u) := \{v \mid v \in [\![p]\!]^\rho_t(u) \text{ and } \rho(x) \neq \lambda^a_t(v)\}.$$

So, $p$ is contained in $q$ w.r.t. $\rho$, denoted $p \subseteq_\rho q$, iff $[\![p]\!]^\rho_t(\varepsilon) \neq \emptyset$ implies $[\![q]\!]^\rho_t(\varepsilon) \neq \emptyset$.

**Theorem 13.** *(a) Containment testing for* $\mathrm{XP}(/, //, [\,], |, *, vars, \neq)$ *under the XPath semantics of variables is in* PSPACE.

*(b) Containment testing for* $\mathrm{XP}(/, //, |, *, vars, \neq)$ *under the XPath semantics of variables is* PSPACE*-hard.*

*Proof.* (sketch)

(a) For the upper bound, we basically show that the problem can be reduced to the case without variables. Let $p$ and $q$ be two expressions with variables $\{x_1, \ldots, x_k\}$. Let us first consider only variable assignments which assign a different value to each variable. Hence, for an attribute $a$ of a node $u$ of a tree $t$ it is only relevant whether the value of $a$ is equal to (exactly) one of the $x_i$ or whether it is different from all of them. Let $t$ be a tree such that $t \models p$ and $t \not\models q$ under assignment $\rho$. We add, for each attribute $a$ of a node $v$, a new child of $v$ labelled by $a$ which itself has a child which is labeled by one of $x_1, \ldots, x_k$ or with `none`. So, if the value of $a$ is $\rho(x_i)$ in $t$, for some $i$,

then it is labelled $x_i$, and `none` if this is not the case, for all $i$. Let us call the resulting tree $t'$. In $p$ we replace each $\$x_i = @a$ by $a/x_i$ and each $\$x_i \neq @a$ by $a/(x_1 \mid \cdots \mid x_{i-1} \mid x_{i+1} \mid \cdots \mid x_k \mid \texttt{none})$. We call the resulting pattern $p'$. Finally we construct $q'$ from $q$ in the same way. It is easy to see that, for each $u$ in $t$: $[\![p]\!]^\rho_t(u) = [\![p']\!]_{t'}(u)$, where we identify the original nodes of $t'$ with their counterparts in $t$.

We conclude that if $\rho$ is a variable assignment with pairwise different values, then $p \subseteq_\rho q$ if and only if $p' \subseteq q'$ on all trees of the form $t'$. Note that these trees have partially a restriction to a finite alphabet. By using a similar proof as for Theorem 6, it can be shown that this test can be done in PSPACE.

The algorithm now cycles through all possible equality types of the variable assignment $\rho$. If, for a particular equality type, two variables get the same value then one of them is replaced by the other in $p$ and in $q$. Hence, we get possibly fewer variables which are again pairwise different and we can apply the above algorithm. The resulting algorithm is in PSPACE.

(b) We use basically the same construction as in Theorem 6. Let $D = \{\sigma_1, \ldots, \sigma_k\}$ be the alphabet used in that construction and assume w.l.o.g. that $k = 2^l$, for some $l$. We use attributes $a_1, \ldots, a_l$ and one variable $x$ to encode the symbols of $D$. E.g., if all $a_i$ of a node $v$ have the same value as $x$ we consider it as labelled with $\sigma_1$. If the value of $a_l$ is different from that of $x$ but all other $a_i$ have the value of $x$, for some $v$ then we interpret this as symbol $\sigma_2$ and so on. In this way, the $k$ symbols correspond to the $k$ different equality types of attributes relative to $x$. In the expressions $p$ and $q$ the element tests are replaced by the wildcard symbol together with the respective attribute comparisons. □

Deutsch and Tannen considered a different semantics which does not assume an external variable binding but rather allows a choice of values for the variables that makes the expression match. More formally, $[\![p]\!]_t(u)$ is defined as the set $\{v \in [\![p]\!]^\rho_t(u) \mid \rho : X \to \mathbf{D}\}$. In particular, for Boolean patterns this means that a tree $t$ matches a pattern $p$ under the semantics of Deutsch and Tannen if there exists a variable assignment $\rho$ such that $t$ matches $p$ relative to $\rho$. We will refer to this sematics as the *existential semantics*.

Deutsch and Tannen showed that containment of XP(/, //, [ ], ∗, |, vars)-expressions under existential semantics is $\Pi^P_2$-complete [10] (Theorem 2.3 and 3.3). Further, they show that containment of XP(/, //, [ ], vars)-expressions is coNP-complete.

Our main results about the existential semantics are the following. When $\neq$ is added to XP(/, //, [ ], ∗, |, vars), then containment is undecidable. However, when $\neq$ is added but ∗ is removed, then containment remains in $\Pi^P_2$. Hardness follows immediately as containment of conjunctive queries (CQs) with inequalities is already $\Pi^P_2$-hard and containment of CQs is reducible to XPath containment.

The results of this section are summarized in Table 2.

**Theorem 14.** *Containment testing for* XP(/, //, [ ], |, *vars*, $\neq$) *under existential semantics is in* $\Pi^P_2$.

**Table 2.** The complexity of containment in the presence of data values under existential semantics. Under XPath semantics the complexity for the full fragment is PSPACE.

| / | // | [] | ∗ | \| | vars | ≠ | complexity |
|---|---|---|---|---|---|---|---|
| + | + | + | | | + | | CONP-complete [10] |
| + | + | | | | + | | CONP-complete |
| + | + | + | + | + | + | | $\Pi_2^p$-complete [10] |
| + | | + | | | + | + | $\Pi_2^p$-complete |
| + | + | + | | + | + | + | $\Pi_2^p$-complete |
| + | + | + | + | + | + | + | undecidable |

*Proof.* (sketch) We show that

(a) $p \not\subseteq q$ if and only if there is a tree $t$ of polynomial size in $|p| + |q|$ such that $t \models p$ but $t \not\models q$, and
(b) Whether $t \models p$ can be tested in NP.

Hence, the algorithm *Guess a tree $t$ of polynomial size and check that $t \models p$ but $t \not\models q$* is a $\Sigma_2$-algorithm for the complement of containment testing.

We omit the proof of (a). To show (b), we remark that whether $t \models p$ for a pattern $p$ in XP$(/, //, [\,], |, \text{vars}, \neq)$ can be tested as follows. First, a disjunct $p_i$ of the disjunctive normal form of $p$ is guessed. Next, a homomorphism from $p_i$ to $t$ and a value assignment for the variables of $p_i$ are guessed (with values $\leq |p_i|$) and it is checked whether all conditions hold. □

A proof of the next theorem is again a reduction from PCP and is omitted due to space restrictions.

**Theorem 15.** *Containment testing for XP(/,//, [ ],∗,|,vars,≠) under existential semantics is undecidable.*

## 6 Discussion

We have studied the complexity of the containment problem for a large class of XPath patterns. In particular, we considered disjunction, DTDs and variables. Unfortunately, the complexity of almost all decidable fragments lies between CONP and EXPTIME. On the other hand, the size of XPath expressions is rather small. As pointed out, Deutsch and Tannen, and Moerkotte already obtained undecidability results for XPath containment. We added two more: presence of node-set equality and modest negation or variables with the existential semantics. It would be interesting to have a precise classification of which combination of features makes the problem undecidable. In a next step, also navigation along the other axes of XPath should be investigated.

**Acknowledgment.** We thank Stijn Vansummeren for comments on a previous version of this paper. We thank the anonymous referees for valuable suggestions.

## References


1. M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. ICDT 2003, this volume.
2. G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.
3. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simeon, and M. Stefanascu. XQuery 1.0: An XML query language. `http://www.w3.org/TR/xquery/`, 2002.
4. B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.
5. J. Clark. XML Path Language (XPath). `http://www.w3.org/TR/xpath`.
6. James Clark. XSL transformations version 1.0. http://www.w3.org/TR/WD-xslt, August 1999.
7. World Wide Web Consortium. XML schema. `http://www.w3.org/XML/Schema`.
8. S. DeRose, E. Maler, and R. Daniel. XML pointer language (XPointer) version 1.0. `http://www.w3.org/TR/xptr/`, 2001.
9. S. DeRose, E. Maler, and D. Orchard. XML linking language (XLink) version 1.0. `http://www.w3.org/TR/xlink/`, 2001.
10. A. Deutsch and V. Tannen. Containment and integrity constraints for xpath. In Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Dan Suciu, editors, *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB 2001)*, number 45 in CEUR Workshop Proceedings, 2001.
11. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, 2002.
12. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proc. of 28th Conf. on VLDB*, 2002.
13. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
14. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 65–76, 2002.
15. G. Moerkotte. Incorporating XSL processing into database engines. In *Proc. of 28th Conf. on VLDB*, 2002.
16. F. Neven. Automata, logic, and XML. In J. C. Bradfield, editor, *CSL*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.
17. F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3), 2002.
18. F. Neven and T. Schwentick. Automata- and logic-based pattern languages for tree-structured data. Unpublished, 2001.
19. F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.
20. G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41(2–3):305–318, 1985.
21. P. T. Wood. Containment for XPath fragments under DTD constraints. ICDT 2003, this volume.
22. P. T. Wood. Minimising simple XPath expressions. WebDB informal proceedings, 2001.
23. P. T. Wood. On the equivalence of XML patterns. In Lloyd et al., editor, *Computational Logic – CL 2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 1152–1166. Springer, 2000.

# Decidable Containment of Recursive Queries

Diego Calvanese[1], Giuseppe De Giacomo[1], and Moshe Y. Vardi[2]

[1] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it,
http://www.dis.uniroma1.it/~lastname/
[2] Department of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu
http://www.cs.rice.edu/~vardi/

**Abstract.** One of the most important reasoning tasks on queries is checking containment, i.e., verifying whether one query yields necessarily a subset of the result of another one. Query containment, is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering. Containment is undecidable in general for Datalog, the fundamental language for expressing recursive queries. On the other hand, it is known that containment between monadic Datalog queries and between Datalog queries and unions of conjunctive queries are decidable. It is also known that containment between unions of conjunctive two-way regular path queries (UC2RPQs), which are queries used in the context of semistructured data models containing a limited form of recursion in the form of transitive closure, is decidable. In this paper we combine the automata-theoretic techniques at the base of these two decidability results to show that containment of Datalog in UC2RPQs is decidable in 2EXPTIME.

## 1 Introduction

Querying is the fundamental mechanism for extracting information from a database. The basic reasoning task associated to querying is query answering, which amounts to computing the information to be returned as result of a query. There are, however, other reasoning services involving queries that data and knowledge representation systems should support. One of the most important is checking containment, i.e., verifying whether one query yields necessarily a subset of the result of another one. Query containment, called *subsumption* in AI [1,2], is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering; cf. [3,4,5,6,7,8,9,10,11,12,13]. Thus, it is fair to describe query containment as one of the most fundamental database reasoning tasks.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 330–345, 2003.

Needless to say, query containment is undecidable if we do not limit the expressive power of the query language; it is clearly undecidable for first-order logic. In fact, in knowledge representation suitable query languages have been designed for retaining decidability. The same is true in databases, where the notion of *conjunctive query* is the basic one in the investigation of reasoning about queries [14]. A conjunctive query (CQ) is simply a conjunction of atoms, where each atom is built out from relation symbols and (existentially quantified) variables. Relationally, a CQ is a project-join query. By adding union and recursion to conjunctive queries, one gets *Datalog*, the language of logic programs (known also as Horn-clause programs) without function symbols [15], which is essentially a fragment of fixpoint logic [16,17]. Datalog consists, in a pure way, only of the most fundamental elements of relational queries: join, projection, union, and recursion. With respect to query containment, CQs and Datalog span the spectrum in terms of computational complexity. In [14] it is shown that CQ containment is equivalent to CQ evaluation (NP-complete). (For some extensions, see [18,19,20,21].) On the other hand, it is shown in [22] that containment of Datalog queries is undecidable; the proof is by reduction from the containment problem for context-free grammars.

The most powerful query-containment results for Datalog are given in [23,24, 25]. In [23] it is pointed out that tree-automata techniques can be used to prove the decidability of query containment for *monadic* Datalog, where rule heads use a single variable (which means that intermediate result of the query, as well as the final one, are sets of data elements). The other results apply to the relationship between Datalog and non-recursive Datalog (non-recursive Datalog queries are in essence unions of conjunctive queries). In [24] it is shown that checking containment of nonrecursive Datalog queries in Datalog queries is decidable in exponential time. In [25] (see also [21]) it is shown, using tree-automata techniques, that containment of Datalog queries in nonrecursive Datalog queries is decidable in triply exponential time. When the non-recursive query is represented, via unfolding, as a union of CQs, the complexity is doubly exponential, rather than triple exponential. (These bounds are known to be optimal, see [26, 4] for studies of special cases and some extensions.)

In this paper we address the problem of query containment in the context of semistructured data models. Our goal is to capture the essential features found in databases, both traditional and semistructured, as well as knowledge bases in semantic networks, conceptual graphs, and description logics. For this purpose, we conceive a database as an edge-labeled graph, where nodes represent objects, and a labeled edge between two nodes represents the fact that the binary relation denoted by the label holds for the objects. This model captures data expressed using XML-like languages [27,28] and is accepted as a standard model for semistructured data [29,30].

In this framework, a basic querying mechanism is the one of *regular path queries* (RPQ) [29,31,32], which ask for all pairs of objects that are connected by a path conforming to a regular expression. Regular path queries are extremely useful for expressing complex navigations in a graph. In particular, union and

transitive closure are crucial when we do not have a complete knowledge of the structure of the database. In our regular path queries, we include also the *inverse* operator, which enables us to navigate edges backwards [29,7], for example, from a child to its parent. We denote these queries by 2RPQs (two-way regular path queries). Using 2RPQs as the basic querying mechanism, one can construct *conjunctive 2-way regular path queries* (C2RPQs), which enables us to perform joins and projections over 2RPQs. C2RPQs are the basic building blocks for querying semistructured data [33,13,31]. The containment problem for C2RPQs (actually for UC2RPQs, unions of such C2RPQs) was studied in [34] (see also [33]), where it was shown, using two-way automata, to be EXPSPACE-complete.

The notable fact about the decidability of containment for C2RPQs is that C2RPQs are a fragment of recursive Datalog, due to the transitive closure operator. Thus, the result in [33,34] is the first decidability result for containment of non-monadic recursive Datalog queries. The fact that automata-theoretic techniques are used both in [25] and in [34] suggests that perhaps the two decidability results can be combined. We show here that this is indeed the case by proving the decidability of the containment of Datalog queries in UC2RPQs (which, implies the known decidability result for containment of UC2RPQs). The automata-theoretic techniques combine tree automata with two-way automata; we use alternating two-way tree automata [35]. The upper bound is doubly exponential time, just as in [25], which we conjecture is optimal (see Conclusions).

## 2 Databases and Queries

We consider a *semistructured database* (DB) $\mathcal{G}$ as an edge-labeled graph $(\mathcal{D}, \mathcal{E})$, where $\mathcal{D}$ is the set of nodes, and $\mathcal{E}$ is the set of edges labeled with elements of an alphabet $\Delta$. A node represents an object, and an edge between nodes $d_1$ and $d_2$ labeled $e$, denoted $e(d_1, d_2)$, represents the fact that the binary relation $e$ holds for the pair $(d_1, d_2)$.

The basic querying mechanism on a DB is that of *regular path queries* (RPQs). An RPQ $E$ is expressed as a regular expression or a finite automaton, and computes the set of pairs of nodes of the DB connected by a path that conforms to the regular language $L(E)$ defined by $E$. We consider unions of conjunctive 2-way regular path queries (UC2RPQs) [34], which extend regular path queries with the possibility to traverse edges backwards, with conjunctions and variables, and with union.

Formally, Let $\Delta$ be a set of binary relation symbols, and let $\Delta^{\pm} = \Delta \cup \Delta^{-}$, with $\Delta^{-} = \{e^{-} \mid e \in \Delta\}$. Intuitively, $e^{-}$ denotes the inverse of the binary relation $e$. If $r \in \Delta^{\pm}$, then we use $r^{-}$ to mean the *inverse* of the relation $r$, i.e., if $r$ is $e$, then $r^{-}$ is $e^{-}$, and if $r$ is $e^{-}$, then $r^{-}$ is $e$.

*2-way regular path queries* (2RPQs) are expressed by means of regular expressions or finite word automata over $\Delta^{\pm}$. Thus, in contrast with RPQs, 2RPQs may use also the inverse $e^{-}$ of $e$, for each $e \in \Delta$. When evaluated over a DB $\mathcal{G}$, a 2RPQ $E$ computes the set $E(\mathcal{G})$ of pairs of nodes $(d_0, d_q)$ such that $r_1(d_0, d_1), r_2(d_1, d_2), \ldots, r_q(d_{q-1}, d_q)$ hold in $\mathcal{G}$ and $r_1 r_2 \cdots r_q$ is in the reg-

ular language $L(E)$ defined by $E$. Observe that, when $q = 0$, we have that $r_1 r_2 \cdots r_q = \varepsilon$ and $d_0 = d_q$.

*Conjunctive 2-way regular path queries* (C2RPQs) are conjunctions of atoms, where each atom specifies that one 2RPQ holds between two variables. More precisely a C2RPQ $\gamma$ of *arity* $n$ is a formula of the form

$$Q(x_1, \ldots, x_n) \leftarrow E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$$

where $x_1, \ldots, x_n, y_1, y_1' \ldots, y_m, y_m'$ range over a set $\{u_1, ..., u_k\}$ of variables, each $x_i$, called a *distinguished variable*, is one of $y_1, y_1' \ldots, y_m, y_m'$, and $E_1, \ldots, E_m$ are 2RPQs. The *answer set* $\gamma(\mathcal{G})$ to a C2RPQ $\gamma$ over a DB $\mathcal{G} = (\mathcal{D}, \mathcal{E})$ is the set of tuples $(d_1, \ldots, d_n)$ of nodes of $\mathcal{G}$ such that there is a total mapping $\sigma$ from $\{u_1, \ldots, u_k\}$ to $\mathcal{D}$ with $\sigma(x_i) = d_i$ for every distinguished variable $x_i$ of $\gamma$, and $(\sigma(y), \sigma(y')) \in E(\mathcal{G})$ for every conjunct $E(y, y')$ in $\gamma$.

Finally, a *union of conjunctive 2-way regular path queries* (UC2RPQ) of arity $n$ has the form $\cup_i \gamma_i$, where each $\gamma_i$ is a C2RPQ of arity $n$. The answer set to a UC2RPQ $\Gamma = \cup_i \gamma_i$ over a DB $\mathcal{G}$ is simply $\Gamma(\mathcal{G}) = \cup_i \gamma_i(\mathcal{G})$. Notice that traditional *conjunctive queries* (resp., unions of conjunctive queries) (cf. [15]) are just a special case of C2RPQs (resp., UC2RPQ) in which each 2RPQ in an atom is simply a relation symbol.

A Datalog program consists of a set of Horn rules. A *(Horn) rule* is a first order material implication between a head and a body, where the head consists of a single atom, and the body consists of a conjunction of atoms. Each atom is a formula of the form $R(x_1, \ldots, x_n)$ where $R$ is a predicate symbol and $x_1, \ldots, x_n$ are variables. All variables are implicitly universally quantified outside the rule. The predicates that occur in heads of rules are called *intensional* (IDB) predicates. The rest of the predicates are called *extensional* (EDB) predicates. Since we consider Datalog programs that are evaluated over a semistructured database, the EDB predicates have to be among the predicates in $\Delta$, which are all binary. Observe, however, that IDB predicates, which are not in $\Delta$, may be of arbitrary arity.

Let $\Pi$ be a Datalog program. Let $Q^i_\Pi(\mathcal{G})$ be the collection of facts about an IDB predicate $Q$ that can be deduced from a database $\mathcal{G}$ by at most $i$ applications of the rules in $\Pi$, and let $Q^\infty_\Pi(\mathcal{G})$ be the collection of facts about $Q$ that can be deduced from $\mathcal{G}$ by any number of applications of the rules in $\Pi$, that is,

$$Q^\infty_\Pi(\mathcal{G}) = \bigcup_{i \geq 0} Q^i_\Pi(\mathcal{G})$$

We say that a Datalog program $\Pi$ with goal predicate $Q$ is *contained* in a UC2RPQ $\Gamma$ if $Q^\infty_\Pi(\mathcal{G}) \subseteq \Gamma(\mathcal{G})$ for every database $\mathcal{G}$.

## 3 Containment of Datalog in Unions of Conjunctive Queries

A *containment mapping* from a conjunctive query $\psi$ to a conjunctive query $\varphi$ is a renaming of variables subject to the following constraints: (a) every distinguished variable must map to itself, and (b) after renaming, every literal in $\psi$ must be among the literals of $\varphi$. It is well known that containment of conjunctive queries can be characterized in terms of containment mappings (cf. [15]). In fact this characterization has been extended in [19] to unions of conjunctive queries, and holds also for infinite unions.

**Theorem 1 ([19]).** *Let $\Phi = \cup_i \varphi_i$ and $\Psi = \cup_i \psi_i$ be (possibly infinite) unions of conjunctive queries. Then $\Phi$ is contained in $\Psi$ (i.e., $\Phi(\mathcal{G}) \subseteq \Psi(\mathcal{G})$ for every database $\mathcal{G}$) if and only if each $\varphi_i$ is contained in some $\psi_j$ , i.e., there is a containment mapping from $\psi_j$ to $\varphi_i$.*

As for containment of Datalog in (unions) of conjunctive queries, it is known (cf. [36,37]) that the relation defined by an IDB predicate in a Datalog program $\Pi$, i.e., $Q_\Pi^\infty(\mathcal{G})$, can be defined by an *infinite* union of conjunctive queries. That is, for each IDB predicate $Q$ there is an infinite sequence $\varphi_0, \varphi_1, \ldots$ of conjunctive queries such that, for every database $\mathcal{G}$, we have $Q_\Pi^\infty(\mathcal{G}) = \bigcup_{i=0}^\infty \varphi_i(\mathcal{G})$. The $\varphi_i$'s are called the *expansions* of $Q$. In [25], expansions of a Datalog program $\Pi$ are described in terms of so-called *expansion trees*, in which each node is labeled with an instance of a rule of $\Pi$. We call head and body of a node the head and the body of the rule labeling the node, respectively. In an expansion tree for an IDB predicate $Q$, the root is labeled by a rule whose head is a $Q$-atom. If a node $g$ is labeled by a rule instance

$$R(\mathbf{t}) \leftarrow R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

where the IDB atoms in the body of the rule are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$, then $g$ has children $g1, \ldots, g\ell$ labeled with rule instances whose heads are respectively the atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$. In particular, if all atoms in the body of $g$ are EDB atoms, then $g$ must be a leaf. The query corresponding to an expansion tree is the conjunction of all EDB atoms in the nodes of the tree, with the variables in the head of the root as the free variables. Thus, we can view an expansion tree $\tau$ as a conjunctive query. Let $\mathit{trees}(Q, \Pi)$ denote the set of expansion trees for an IDB predicate $Q$ in $\Pi$. (Note that $\mathit{trees}(Q, \Pi)$ is an infinite set.) Then for every database $\mathcal{G}$, we have

$$Q_\Pi^\infty(\mathcal{G}) = \bigcup_{\tau \in \mathit{trees}(Q,\Pi)} \tau(\mathcal{G})$$

It follows that $\Pi$ is contained in a conjunctive query $\varphi$ if there is a containment mapping from $\varphi$ to each expansion tree $\tau$ in $\mathit{trees}(Q, \Pi)$, i.e., a mapping, which maps distinguished variables to distinguished variables and maps the atoms of $\varphi$ to atoms in the bodies of rules labeling nodes of $\tau$.

Unfortunately, the number of variables, and hence the number of node labels in expansion trees is not bounded, and thus expansion trees are not directly suited for an automata-theoretic approach to containment. In [25], the notion of *proof tree* is introduced, with the idea of describing expansion trees using a finite number of labels. The number of labels is bound by bounding the set of variables that can occur in labels of nodes in the tree. If $r$ is a rule of a Datalog program $\Pi$, then let $num\_var(r)$ be the number of variables occurring in IDB atoms in $r$ (head or body). Let $num\_var(\Pi)$ be twice the maximum of $num\_var(r)$ for all rules $r$ in $\Pi$. Let $var(\Pi)$ be the set $\{x_1, \ldots, x_{num\_var(\Pi)}\}$. A *proof tree* for $\Pi$ is simply an expansion tree for $\Pi$ all of whose variables are from $var(\Pi)$. We denote the set of proof trees for a predicate $Q$ of a Datalog program $\Pi$ by $p\_trees(Q, \Pi)$.

A proof tree represents an expansion tree where variables are re-used. In other words, the same variable is used to represent a set of distinct variables in the expansion tree. Intuitively, to reconstruct an expansion tree for a given proof tree, we need to distinguish among occurrences of variables. Let $g_1$ and $g_2$ be nodes in a proof tree $\tau$, with a lowest common ancestor $g_0$, and let $\mathsf{x}_1$ and $\mathsf{x}_2$ be occurrences, in $g_1$ and $g_2$, respectively, of a variable $x$. We say that $\mathsf{x}_1$ and $\mathsf{x}_2$ are *connected* in $\tau$ if the head of every node, except perhaps for $g_0$, on the simple path connecting $g_1$ and $g_2$ has an occurrence of $x$. We say that an occurrence $\mathsf{x}$ of a variable $x$ in $\tau$ is a *distinguished occurrence* if it is connected to an occurrence of $x$ in the head of the root of $\tau$.

We want to define containment mappings from conjunctive queries to proof trees such that there is a containment mapping from a conjunctive query to a proof tree if and only if there is a containment mapping from the conjunctive query to the expansion corresponding to the proof tree. To do so, we need to force a variable in the conjunctive query to map to a unique variable in the expansion corresponding to the proof tree. A *strong containment mapping* from a conjunctive query $\varphi$ to a proof tree $\tau$ is a containment mapping $h$ from $\varphi$ to $\tau$ with the following properties:

- $h$ maps distinguished occurrences in $\varphi$ to distinguished occurrences in $\tau$, and
- if $\mathsf{x}_1$ and $\mathsf{x}_2$ are two occurrences of a variable $x$ in $\varphi$, then the occurrences $h(\mathsf{x}_1)$ and $h(\mathsf{x}_2)$ in $\tau$ are connected.

The following characterization of containment of a union of conjunctive queries in a Datalog program was shown in [25].

**Theorem 2 ([25]).** *Let $\Pi$ be a Datalog program with goal predicate $Q$, and let $\Phi = \cup_i \varphi_i$ be a (possibly infinite) union of conjunctive queries. Then $\Pi$ is contained in $\Phi$ if and only if for every proof tree $\tau \in p\_trees(Q, \Pi)$ there is a strong containment mapping from some $\varphi_i$ to $\tau$.*

The above theorem is shown in [25] for *finite* unions of conjunctive queries only. However, it is easy to see that the proof carries through also for infinite unions.

Notice that Theorem 2 by itself does not provide decidability of containment of Datalog in (possibly infinite) unions of conjunctive queries, since one needs a

method to check the existence of a strong containment mapping. Undecidability of containment between Datalog queries [22] shows that such a method will not exist in general for (infinite) unions that are expansions of Datalog programs. However, in [25] the above result is exploited to show that containment of a Datalog query in a finite union of conjunctive queries is in 2EXPTIME (and in fact 2EXPTIME-complete).

To exploit Theorem 2 for containment of Datalog queries in UC2RPQs, we need to characterize the problem in terms of containment between Datalog and (infinite) unions of conjunctive queries. An *expansion* of a C2RPQ

$$Q(x_1, \ldots, x_n) \leftarrow E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$$

is a CQ of the form

$$\begin{aligned} Q(x_1, \ldots, x_n) \leftarrow\ & r_1^1(y_1, z_1^1), r_1^2(z_1^1, z_1^2), \ldots, r_1^{n_1}(z_1^{n_1-1}, y_1'), \\ & \vdots \\ & r_m^1(y_m, z_m^1), r_m^2(z_m^1, z_m^2), \ldots, r_m^{n_m}(z_m^{n_m-1}, y_m') \end{aligned}$$

where, for each $i \in \{1, \ldots, m\}$, we have that $n_i \geq 0$, that $r_i^1 \cdots r_i^{n_i} \in L(E_i)$, and that all variables $z_i^j$ are pairwise distinct. Observe that, when $n_i = 0$, we have that $r_i^1 \cdots r_i^{n_i} = \varepsilon$, and $r_i^1(y_i, z_i^1), r_i^2(z_i^1, z_i^2), \ldots, r_i^{n_i}(z_i^{n_i-1}, y_i')$ becomes simply $y_i = y_i'$. Notice that, due to transitive closure, a C2RPQ has in general an infinite number of expansions.

The following lemma is an easy consequence of Theorem 2 and of the semantics of UC2RPQs.

**Lemma 1.** *Let $\Pi$ be a Datalog program with goal predicate $Q$, and let $\Gamma = \cup_i \gamma_i$ be a finite union of C2RPQs $\gamma_i$. Then $\Pi$ is contained in $\Gamma$ if and only if for every proof tree $\tau \in p\_trees(Q, \Pi)$ there is a $\gamma_i$ and an expansion $\varphi$ of $\gamma_i$ such that there is a strong containment mapping from $\varphi$ to $\tau$.*

We show how to check this condition using tree automata.

## 4 Two-Way Alternating Tree Automata

We present the basic notions on automata used in the rest of the paper. We assume familiarity with the standard notions of (one-way) word automata (1NFAs) and (one-way) nondeterministic tree automata (1NTAs), and concentrate on two-way alternating tree automata (2ATAs).

Trees are represented as prefix closed finite sets of words over $\mathbb{N}$ (the set of positive natural numbers). Formally, a *tree* $T$ is a finite subset of $\mathbb{N}$, such that if $g \cdot c \in T$, where $g \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $g \in T$ and if $c > 1$ then also $g \cdot (c-1) \in T$. The elements of $T$ are called *nodes*, and for every $g \in T$, the nodes $g \cdot c \in T$, with $c \in \mathbb{N}$, are the *successors* of $g$. By convention we take $g \cdot 0 = g$, and $g \cdot c \cdot (-1) = g$. By definition, the empty sequence $\varepsilon$ is a member of every tree, and is called the *root*. Note that $\varepsilon \cdot -1$ is undefined. The *branching degree* $d(g)$

of a node $g$ denotes the number of successors of $g$. If the branching degree of all nodes of a tree is bounded by $k$, we say that the tree has branching degree $k$. Given a finite alphabet $\Sigma$, a *$\Sigma$-labeled tree* $\tau$ is a pair $(T, V)$, where $T$ is a tree and $V : T \to \Sigma$ maps each node of $T$ to an element of $\Sigma$. $\Sigma$-labeled trees are often referred to as *trees*, and if $\tau = (T, V)$ is a (labeled) tree and $g$ is a node of $T$, we use $\tau(g)$ to denote $V(g)$.

Two-way alternating tree automata (2ATAs) [35,23], are a generalization of standard nondeterministic top-down tree automata (1NTAs) [38,39]) with both upward moves and with alternation. Let $\mathcal{B}(I)$ be the set of positive Boolean formulae over $I$, built inductively by applying $\wedge$ and $\vee$ starting from **true**, **false**, and elements of $I$. For a set $J \subseteq I$ and a formula $\varphi \in \mathcal{B}(I)$, we say that $J$ *satisfies* $\varphi$ if and only if, assigning **true** to the elements in $J$ and **false** to those in $I \setminus J$, makes $\varphi$ true. For a positive integer $k$, let $[k] = \{-1, 0, 1, \ldots, k\}$. A *two-way alternating tree automaton* (2ATA) over an alphabet $\Sigma$ running over trees with branching degree $k$, is a tuple $\mathbf{A} = (\Sigma, S, \delta, s_0, F)$, where $S$ is a finite set of states, $\delta : S \times \Sigma \to \mathcal{B}([k] \times S)$ is the transition function, $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of final states. The transition function maps a state $s \in S$ and an input letter $\sigma \in \Sigma$ to a positive Boolean formula over $[k] \times S$. Intuitively, if $\delta(s, \sigma) = \varphi$, then each pair $(c, s')$ appearing in $\varphi$ corresponds to a new copy of the automaton going to the direction suggested by $c$ and starting in state $s'$.

A run $\nu$ of a 2ATA $\mathbf{A}$ over a labeled tree $\tau = (T, V)$ is a labeled tree $(T_\nu, V_\nu)$ in which every node is labeled by an element of $T \times S$. A node $f$ of $T_\nu$ labeled by $(g, s)$ describes a copy of $\mathbf{A}$ that is in the state $s$ and reads the node $g$ of $\tau$. The labels of adjacent nodes have to satisfy the transition function of $\mathbf{A}$. Formally, a run $(T_\nu, V_\nu)$ is a $(T \times S)$-labeled tree satisfying:

1. $\varepsilon \in T_\nu$ and $V_\nu(\varepsilon) = (\varepsilon, s_0)$.
2. Let $f \in T_\nu$, with $V_\nu(f) = (g, s)$ and $\delta(s, V(g)) = \varphi$. Then there is a (possibly empty) set $C = \{(c_1, s_1), \ldots, (c_n, s_n)\} \subseteq [k] \times S$ such that:
   - $C$ satisfies $\varphi$ and
   - for all $i \in \{1, \ldots, n\}$, we have that $f{\cdot}i \in T_\nu$, $g{\cdot}c_i$ is defined, and $V_\nu(f{\cdot}i) = (g{\cdot}c_i, s_i)$.

A run $\nu = (T_\nu, V_n u)$ on a tree $\tau$ is *accepting* if, whenever a leaf of $T_\nu$ is labeled by $(g, s)$, then $s \in F$. $\mathbf{A}$ *accepts* a labeled tree $\tau$ if it has an accepting run on $\tau$. The set of trees accepted by $\mathbf{A}$ is denoted $\mathcal{T}(\mathbf{A})$. The *nonemptiness* problem for tree automata consists in deciding, given a tree automaton $\mathbf{A}$, whether $\mathcal{T}(\mathbf{A})$ is nonempty.

As shown in [23], 2ATAs can be converted to complementary 1NTAs with only a single exponential blowup. Moreover, it is straightforward to see that one can construct a 2ATA of polynomial size accepting the finite union of the languages accepted by $n$ 2ATAs.

**Proposition 1 ([23]).** *Given a 2ATA $\mathbf{A}$ over an alphabet $\Sigma$, there is a 1NTA $\overline{\mathbf{A}}$ of size exponential in the size of $\mathbf{A}$ such that $\overline{\mathbf{A}}$ accepts a $\Sigma$-labeled tree $\tau$ if and only if $\tau$ is rejected by $\mathbf{A}$.*

**Proposition 2.** *Given $n$ 2ATAs $\mathbf{A}_1, \ldots, \mathbf{A}_n$ over an alphabet $\Sigma$, there is a 2ATA $\mathbf{A}_\cup$ of size polynomial in the sum of the sizes of $\mathbf{A}_1, \ldots, \mathbf{A}_n$ such that $\mathcal{T}(\mathbf{A}_\cup) = \mathcal{T}(\mathbf{A}_1) \cup \cdots \cup \mathcal{T}(\mathbf{A}_n)$.*

We make also use of the following standard results for 1NTAs.

**Proposition 3 ([40]).** *Given 1NTAs $\mathbf{A}_1$ and $\mathbf{A}_2$ over an alphabet $\Sigma$, there is a 1NTA $\mathbf{A}_\cap$ of size polynomial in the size of $\mathbf{A}_1$ and $\mathbf{A}_2$ such that $\mathcal{T}(\mathbf{A}_\cap) = \mathcal{T}(\mathbf{A}_1) \cap \mathcal{T}(\mathbf{A}_2)$.*

**Proposition 4 ([38,39]).** *The nonemptiness problem for 1NTAs is decidable in polynomial time.*

## 5 Containment of Datalog in Unions of C2RPQs

The main feature of proof trees is the fact that the number of possible labels is finite; it is actually exponential in the size of $\Pi$. Because the set of labels is finite, the set of proof trees $p\_trees(Q, \Pi)$, for an IDB predicate $Q$ in a program $\Pi$, can be described by a tree automaton.

**Theorem 3 ([25]).** *Let $\Pi$ be a Datalog program with a goal predicate $Q$. Then there is a 1NTA $\mathbf{A}^{p\_trees}_{Q,\Pi}$, whose size is exponential in the size of $\Pi$, such that $\mathcal{T}(\mathbf{A}^{p\_trees}_{Q,\Pi}) = p\_trees(Q, \Pi)$.*

The automaton $\mathbf{A}^{p\_trees}_{Q,\Pi} = (\Sigma, \mathcal{I} \cup \{accept\}, \mathcal{I}_Q, \delta, \{accept\})$ defined in [25] is as follows. The state set $\mathcal{I}$ is the set of all IDB atoms with variables among $var(\Pi)$. The start-state set $\mathcal{I}_Q$ is the set of all atoms $Q(\mathbf{s})$, where the variables of $\mathbf{s}$ are in $var(\Pi)$. The alphabet is $\Sigma = \mathcal{I} \times \mathcal{R}$, where $\mathcal{R}$ is the set of instances of rules of $\Pi$ over $var(\Pi)$. The transition function $\delta$ is constructed as follows. Let $\varrho$ be the body of a rule instance in $\mathcal{R}$

$$R(\mathbf{t}) \;\leftarrow\; R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

– If the IDB atoms in $\varrho$ are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$, then there is a transition[1]

$$\langle 1, R_{i_1}(\mathbf{t}^{i_1})\rangle \wedge \cdots \wedge \langle \ell, R_{i_\ell}(\mathbf{t}^{i_\ell})\rangle \;\in\; \delta(R(\mathbf{t}), (R(\mathbf{t}) \leftarrow \varrho))$$

– If all atoms in $\varrho$ are EDB atoms, then there is a transition

$$\langle 0, accept\rangle \;\in\; \delta(R(\mathbf{t}), (R(\mathbf{t}) \leftarrow \varrho))$$

It is easy to see that the number of states and transitions in $\mathbf{A}^{p\_trees}_{Q,\Pi}$ is exponential in the size of $\Pi$.

We now show that strong containment of proof trees in a C2RPQ can be checked by tree automata as well. Let $\Pi$ be a Datalog program with binary

[1] For uniformity, we use the notation of 2ATAs to denote the transitions of 1NTAs.

EDB predicates in $\Delta$ and with goal predicate $Q$, and let $\gamma$ be a C2RPQ over $\Delta^{\pm}$ of the same arity as $Q$. We describe the construction of a 2ATA $\mathbf{A}^{\gamma}_{Q,\Pi}$ that accepts all proof trees $\tau$ in $p\_trees(Q,\Pi)$ such that there is an expansion $\varphi$ of $\gamma$ and a strong containment mapping from $\varphi$ to $\tau$.

We view $\gamma$ as a set of atoms $E(x,y)$, where $E$ is a 1NFA $E = (\Delta^{\pm}, S_E, s_E, \delta_E, f_E)$, with $s_E, f_E \in S_E$, and where, w.l.o.g., $\delta_E$ does not contain $\varepsilon$-transitions. Also, w.l.o.g., we assume that for two distinct atoms $E_1(x_1,y_1)$ and $E_2(x_2,y_2)$, $E_1$ and $E_2$ are distinct automata with disjoint sets of states, i.e., $S_{E_1} \cap S_{E_2} = \emptyset$. For a 1NFA $E$, we use $E^f_s$ to denote the 1NFA identical to $E$, except that $s \in S_E$ and $f \in S_E$ are respectively the initial and final state of $E^f_s$.

Let $V_\gamma$ be the set of variables appearing in the C2RPQ $\gamma$, and $V^+_\gamma = \{\bar{v}^1_E, \bar{v}^2_E \mid E(x,y) \in \gamma\}$, i.e., for each 1NFA $E(x,y) \in \gamma$, $V^+_\gamma$ contains two special variables $\bar{v}^1_E$ and $\bar{v}^2_E$. We denote with $\mathcal{B}$ the set of all sets $\beta$ of atoms, such that $\beta$ contains, for each atom $E(x,y) \in \gamma$, at most one atom $E^f_s(x',y')$, for some $s, f \in S_E$, with $x'$ either $x$ or $\bar{v}^1_E$ and $y'$ either $y$ or $\bar{v}^2_E$. Notice that the size of $\mathcal{B}$ is exponential in the size of $\gamma$. Indeed, let $k$ be the number of atoms in $\gamma$ and let $m$ be an upper bound on the number of states of each 1NFA in $\gamma$. All possible variants of a 1NFA obtained by changing the initial state and/or final state are $m^2$. Hence, the number of possible sets of 1NFAs of at most $k$ elements is $(m^2)^k = 2^{O(m \cdot k)}$.

The automaton $\mathbf{A}^{\gamma}_{Q,\Pi}$ is $(\Sigma, S \cup \{accept\}, S_Q, \delta, \{accept\})$.

- The alphabet $\Sigma$ is $\mathcal{I} \times \mathcal{R}$. Recall that $\mathcal{I}$ is the set of all IDB atoms with variables among $var(\Pi)$, and $\mathcal{R}$ is the set of instances of rules of $\Pi$ over $var(\Pi)$.
- The state set $S$ is the set $\mathcal{I} \times \mathcal{B} \times 2^{V_\gamma \times var(\Pi)} \times 2^{V^+_\gamma \times var(\Pi)}$. The second component represents the collection of automata accepting sequences of atoms that have to be mapped to atoms in the tree $\tau$ accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$, and the third and fourth component contain the set of partial mappings respectively from $V_\gamma$ and $V^+_\gamma$ to $var(\Pi)$.
- The start-state set $S_Q$ consists of all tuples $(Q(\mathbf{s}), \gamma, M_{\gamma,\mathbf{s}}, \emptyset)$, where the variables of $\mathbf{s}$ are in $var(\Pi)$ and $M_{\gamma,\mathbf{s}}$ is a mapping of the distinguished variables of $\gamma$ into the variables of $\mathbf{s}$.

The transition function $\delta$ of $\mathbf{A}^{\gamma}_{Q,\Pi}$ is constructed as follows. Let $\varrho$ be the body of a rule instance in $\mathcal{R}$

$$R(\mathbf{t}) \leftarrow R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

1. There is an "atom mapping" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M'_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if there is an EDB atom $e(a,b)$ among $R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$ and if $\beta'$ coincides with $\beta$, except that one element $E^f_s(x,y)$ in $\beta$ is replaced in $\beta'$ by $E^f_{s'}(x',y)$, and one of the following holds:

- $s' \in \delta_E(s,e)$ and
  - if $x \in V_\gamma$ (i.e., $x$ is a variable of $\gamma$), $M$ maps $x$ to $a$, and $M_+$ does not map $\bar{v}^1_E$, then $x' = \bar{v}^1_E$ and $M'_+ = M_+ \cup \{(\bar{v}^1_E, b)\}$;

- if $x = \bar{v}_E^1 \in V_\gamma^+$ (i.e., $x$ is the first special variable for the 1NFA $E$) and $(\bar{v}_E^1, a) \in M_+$, then $x' = x = \bar{v}_E^1$, and $M'_+ = M_+ \setminus \{(\bar{v}_E^1, a)\} \cup \{(\bar{v}_E^1, b)\}$;

– $s' \in \delta_E(I, e^-)$ and
- if $x \in V_\gamma$ (i.e., $x$ is a variable of $\gamma$), $M$ maps $x$ to $b$, and $M_+$ does not map $\bar{v}_E^1$, then $x' = \bar{v}_E^1$ and $M'_+ = M_+ \cup \{(\bar{v}_E^1, a)\}$;
- if $x = \bar{v}_E^1 \in V_\gamma^+$ (i.e., $x$ is the first special variable for the 1NFA $E$) and $(\bar{v}_E^1, b) \in M_+$, then $x' = x = \bar{v}_E^1$, and $M'_+ = M_+ \setminus \{(\bar{v}_E^1, b)\} \cup \{(\bar{v}_E^1, a)\}$.

Intuitively, an "atom mapping" transition maps the next atom recognized by some 1NFA in $\beta$ to some EDB atom in $\rho$, and modifies $M_+$ accordingly. Note that the variable $x$ (either a variable of $V_\gamma$ or the special variable $\bar{v}_E^1$) must already be mapped (respectively by $M$ or $M_+$) to some variable in the current node of $\tau$.

2. There is a "splitting" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M'_+)\rangle \ \wedge \ \langle 0, (R(\mathbf{t}), \beta'', M, M''_+)\rangle \ \in \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if the following hold:
– $M'_+$ and $M''_+$ coincide with $M_+$, except for the changes described in the following point;
– $\beta$ can be partitioned into $\beta_1$, $\beta_2$, and $\beta_3$; moreover $\beta' = \beta_1 \cup \beta'_3$ and $\beta'' = \beta_2 \cup \beta''_3$, where $\beta'_3$ and $\beta''_3$ are sets of elements that consist of one element for each element $E_s^f(x, y)$ in $\beta_3$, obtained as follows: for some state $s'$ of $E$ and some variable $a \in var(\Pi)$ appearing in $R(\mathbf{t}) \leftarrow \varrho$, one of the following holds:
- $\beta'_3$ contains the element $E_s^{s'}(x, \bar{v}_E^2)$, $\beta''_3$ contains the element $E_{s'}^f(\bar{v}_E^1, y)$, $M'_+$ (re-)maps $\bar{v}_E^2$ to $a$, and $M''_+$ (re-)maps $\bar{v}_E^1$ to $a$;
- $\beta'_3$ contains the element $E_{s'}^f(\bar{v}_E^1, y)$, $\beta''_3$ contains the element $E_s^{s'}(x, \bar{v}_E^2)$, $M'_+$ (re-)maps $\bar{v}_E^1$ to $a$, and $M''_+$ (re-)maps $\bar{v}_E^2$ to $a$;

– $\beta'$ and $\beta''$ can share a variable in $V_\gamma$ only if this variable is in the domain of $M$. (Notice that two occurrences of a special variable in $V_\gamma^+$ shared by $\beta'$ and $\beta''$ are not related to each other.)

A "splitting" transition partitions the atoms in $\beta$ into two parts. The goal is to enable the two parts to be manipulated separately. For example, one part may correspond to those atoms that are intended to be "moved" together to an adjacent node in a future transition, while the other part may correspond to those atoms that are meant to stay together in the current node for further processing, e.g., by further splitting or by mapping to EDB atoms. During splitting, some atoms in $\beta$ may be actually split into two subatoms. The mappings $M$ and $M_+$ have to "bind" together variables that are in common to the two conjuncts of the transition.

3. There is a "moving" transition

$$\langle j, (R_{i_j}(\mathbf{t}^{i_j}), \beta, M, M_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

with $j \in \{-1, 1, \ldots, \ell\}$, where $\ell$ is the number of IDB atoms in $\varrho$, the atom $R_{i_j}(\mathbf{t}^{i_j})$, for $j \in \{1, \ldots, \ell\}$, is the $j$-th IDB atom, $R_{i_{-1}}$ stands for $R$, and $\mathbf{t}^{i_{-1}}$ stands for $\mathbf{t}$, if for all variables that occur in $\beta$ and that are in the domain of either $M$ or $M_+$, their image is in $\mathbf{t}^{i_j}$.

A "moving" transition moves to an adjacent node, and is intended to be applied whenever no next atom can be mapped and no further splitting is possible. Moving is possible only if variables that are both in atoms still to be mapped (and thus in $\beta$) and have already been mapped (and thus are in the domain of either $M$ or $M_+$) can be propagated through the head of the rule where the automaton moves.

4. There is an "equality checking" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if the following hold:
   - $\beta$ can be partitioned into $\beta_0$ and $\beta'$;
   - for all atoms $E_s^f(x, y) \in \beta_0$ we have that
     - $s = f$,
     - $(x, a)$ and $(y, a)$ are in $M \cup M_+$, for some variable $a$ in $\varrho$ or $\mathbf{t}$, i.e., both $x$ and $y$ are in the domain of $M$ or of $M_+$ and they are mapped to the same variable $a$;

   An "equality checking" transition gets rid of those elements in $\beta$ all of whose atoms have already been mapped to atoms in $\tau$. While doing so, it checks that $M$ and $M_+$ are compatible with the equalities induced by such atoms.
5. There is a "mapping extending" transition

$$\langle 0, (R(\mathbf{t}), \beta, M', M_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if $M'$ is a partial mapping that extends $M$.

A "mapping extending" transition adds some variables to the mapping $M$. This may be necessary to be able to apply some other transition that requires certain variables to appear in $M$.
6. There is a "final" transition

$$\langle 0, \mathit{accept}\rangle \ \in \ \delta((R(\mathbf{t}), \emptyset, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

A "final" transition moves to the accepting state whenever there are no further atoms in $\beta$ that have to be processed.

It is easy to see that the number of states and transitions in $\mathbf{A}^{\gamma}_{Q,\Pi}$ is exponential in the size of $\Pi$ and $\gamma$. The following two basic lemmas establish the correctness of the above construction.

**Lemma 2.** *Let $\tau$ be a proof tree in $p\_trees(Q, \Pi)$. If there is an expansion $\varphi$ of $\gamma$ and a strong containment mapping $h$ from $\varphi$ to $\tau$, then $\tau$ is accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$.*

**Lemma 3.** *Let $\tau$ be a proof tree in $p\_trees(Q, \Pi)$. If $\tau$ is accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$, then there is an expansion $\varphi$ of $\gamma$ and a strong containment mapping from $\varphi$ to $\tau$.*

**Theorem 4.** *Let $\Pi$ be a Datalog program with binary EDB predicates in $\Delta$ and with goal predicate $Q$, and let $\Gamma = \cup_i \gamma_i$ be a finite union of C2RPQs $\gamma_i$ over $\Delta^{\pm}$. Then $\Pi$ is contained in $\Gamma$ if and only if*

$$\mathcal{T}(\mathbf{A}_{Q,\Pi}^{p\_trees}) \subseteq \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$$

*Proof.* By Lemma 1, $\Pi$ is contained in $\Gamma$ if and only if for every proof tree $\tau \in p\_trees(Q, \Pi)$ there is a $\gamma_i$ and an expansion $\varphi$ of $\gamma_i$ such that there is a strong containment mapping from $\varphi$ to $\tau$. By Theorem 3 and Lemmas 2 and 3, the latter conditions is equivalent to $\mathcal{T}(\mathbf{A}_{Q,\Pi}^{p\_trees}) \subseteq \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$.

This allows us to establish the main result of the paper.

**Theorem 5.** *Containment of a recursive Datalog program in a UC2RPQ is in 2EXPTIME.*

*Proof.* By Proposition 2, we can construct a 2ATA $\mathbf{A}_{Q,\Pi}^{\Gamma}$, whose size is exponential in the size of $\Pi$ and $\Gamma$, such that $\mathcal{T}(\mathbf{A}_{Q,\Pi}^{\Gamma}) = \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$. By Proposition 1, we can construct a 1NTA $\mathbf{A}_{Q,\Pi}^{\neg\Gamma}$, whose size is doubly exponential in the size of $\Pi$ and $\Gamma$, such that a $\Sigma$-labeled tree is accepted by $\mathbf{A}_{Q,\Pi}^{\neg\Gamma}$ if and only if it is not accepted by $\mathbf{A}_{Q,\Pi}^{\Gamma}$. By Proposition 3, we can construct a 1NTA $\mathbf{A}_{cont}$, whose size is still doubly exponential in the size of $\Pi$ and $\Gamma$, such that $\mathbf{A}_{cont}$ accepts a $\Sigma$-labeled tree if and only if it is accepted by $\mathbf{A}_{Q,\Pi}^{p\_trees}$ but not accepted by any of the $\mathbf{A}_{Q,\Pi}^{\gamma_i}$. By Theorem 4, $\mathbf{A}_{cont}$ is nonempty if and only if $\Pi$ is not contained in $\Gamma$. By Proposition 4, nonemptiness of $\mathbf{A}_{cont}$ can be checked in time polynomial in its size, and hence doubly exponential in the size of $\Pi$ and $\Gamma$. The claim follows.

## 6 Conclusions

We have presented an upper-bound result for containment of Datalog queries in unions of conjunctive regular path queries with inverse (UC2RPQ). This is the most general known decidability result for containment of recursive queries, apart from the result in [23] for monadic Datalog. The class UC2RPQ has several features that are typical of modern query languages for knowledge and data bases. In particular, it is the largest fragment of query languages for XML data [41] for which containment is known to be decidable [34].

The 2EXPTIME upper-bound result shows that adding transitive closure to conjunctive queries does not increase the complexity of query containment with respect to Datalog queries, as it matches the bound obtained in [25] for containment of Datalog queries in union of conjunctive queries. For containment in union of conjunctive queries, the 2EXPTIME bound is shown in [25] to be tight. It is an open question whether our bound here is also tight. The lower bound in [25] is shown using relation symbols of arity up to 8. If that arity can be reduced to 2, then it would follow that our bound here is tight. We conjecture this to be the case. Currently, we have an EXPSPACE lower bound that directly

follows from EXPSPACE-completeness of containment of UC2RPQs [34] (which is a special case of containment of Datalog in UC2RPQs). Observe that containment in the converse direction, as well as equivalence, is undecidable already for RPQs. Indeed, universality of context free grammars can be reduced to containment of RPQs in Datalog, by following the line of the undecidability proof of containment between Datalog queries in [22].

Query containment is typically the first step in addressing various problems of query processing, such as view-based query processing. We predict that the decidability result for containment obtained in this paper would prove useful for a broad range of query processing applications.

**Acknowledgements.** The first and second author were supported in part by MIUR project D2I (Integration, Warehousing and Mining of Heterogeneous Data Sources), by EU Project INFOMIX (Boosting Information Integration) IST-2001-33570, and by EU Project SEWASIE (Semantic Webs and AgentS in Integrated Economies) IST-2001-34825. The third author was supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264.

## References

1. Buchheit, M., Jeusfeld, M.A., Nutt, W., Staudt, M.: Subsumption between queries to object-oriented databases. Information Systems **19** (1994) 33–54 Special issue on Extending Database Technology, EDBT'94.
2. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics. In Brewka, G., ed.: Principles of Knowledge Representation. Studies in Logic, Language and Information. CSLI Publications (1996) 193–238
3. Gupta, A., Ullman, J.D.: Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In: Workshop on Deductive Databases (In conjunction with JICSLP), Washington D.C. (USA) (1992) 195
4. Levy, A.Y., Sagiv, Y.: Semantic query optimization in Datalog programs. In: Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95). (1995) 163–173
5. Chaudhuri, S., Krishnamurthy, S., Potarnianos, S., Shim, K.: Optimizing queries with materialized views. In: Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95), Taipei (Taiwan) (1995)
6. Adali, S., Candan, K.S., Papakonstantinou, Y., Subrahmanian, V.S.: Query caching and optimization in distributed mediator systems. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data. (1996) 137–148
7. Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization technique for unstructured data. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data. (1996) 505–516
8. Motro, A.: Panorama: A database system that annotates its answers to queries with their properties. J. of Intelligent Information Systems **7** (1996)
9. Levy, A.Y., Rousset, M.C.: Verification of knowledge bases: a unifying logical view. In: Proc. of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems, Leuven, Belgium (1997)

10. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description logic framework for information integration. In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98). (1998) 2–13
11. Fernandez, M.F., Florescu, D., Levy, A., Suciu, D.: Verifying integrity constraints on web-sites. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99). (1999) 614–619
12. Friedman, M., Levy, A., Millstein, T.: Navigational plans for data integration. In: Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99), AAAI Press/The MIT Press (1999) 67–73
13. Milo, T., Suciu, D.: Index structures for path expressions. In: Proc. of the 7th Int. Conf. on Database Theory (ICDT'99). Volume 1540 of Lecture Notes in Computer Science., Springer (1999) 277–295
14. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77). (1977) 77–90
15. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading, Massachussetts (1995)
16. Chandra, A.K., Harel, D.: Horn clause queries and generalizations. J. of Logic and Computation **2** (1985) 1–15
17. Moschovakis, Y.N.: Elementary Induction on Abstract Structures. North-Holland Publ. Co., Amsterdam (1974)
18. Aho, A.V., Sagiv, Y., Ullman, J.D.: Equivalence among relational expressions. SIAM J. on Computing **8** (1979) 218–246
19. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. of the ACM **27** (1980) 633–655
20. Klug, A.C.: On conjunctive queries containing inequalities. J. of the ACM **35** (1988) 146–160
21. van der Meyden, R.: The Complexity of Querying Indefinite Information. PhD thesis, Rutgers University (1992)
22. Shmueli, O.: Equivalence of Datalog queries is undecidable. J. of Logic Programming **15** (1993) 231–241
23. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs. In: Proc. of the 20th ACM SIGACT Symp. on Theory of Computing (STOC'88). (1988) 477–490
24. Sagiv, Y.: Optimizing Datalog programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos (1988) 659–698
25. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive datalog programs. J. of Computer and System Sciences **54** (1997) 61–78
26. Chaudhuri, S., Vardi, M.Y.: On the complexity of equivalence between recursive and nonrecursive Datalog programs. In: Proc. of the 13th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'94). (1994) 107–116
27. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0 — W3C recommendation. Technical report, World Wide Web Consortium (1998) Available at `http://www.w3.org/TR/1998/REC-xml-19980210`.
28. Calvanese, D., De Giacomo, G., Lenzerini, M.: Representing and reasoning on XML documents: A description logic approach. J. of Logic and Computation **9** (1999) 295–318
29. Buneman, P.: Semistructured data. In: Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97). (1997) 117–121

30. Florescu, D., Levy, A., Mendelzon, A.: Database techniques for the World-Wide Web: A survey. SIGMOD Record **27** (1998) 59–74
31. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: from Relations to Semistructured Data and XML. Morgan Kaufmann, Los Altos (2000)
32. Abiteboul, S., Vianu, V.: Regular path queries with constraints. J. of Computer and System Sciences **58** (1999) 428–452
33. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). (1998) 139–148
34. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000). (2000) 176–185
35. Slutzki, G.: Alternating tree automata. In: Theoretical Computer Science. Volume 41. (1985) 305–318
36. Maier, D., Ullman, J.D., Vardi, M.Y.: On the foundations of the universal relation model. ACM Trans. on Database Systems **9** (1984) 283–308
37. Naughton, J.F.: Data independent recursion in deductive databases. J. of Computer and System Sciences **38** (1989) 259–289
38. Doner, J.E.: Tree acceptors and some of their applications. J. of Computer and System Sciences **4** (1970) 406–451
39. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second order logic. Mathematical Systems Theory **2** (1968) 57–81
40. Costich, O.L.: A Medvedev characterization of sets recognized by generalized finite automata. Mathematical Systems Theory **6** (1972) 263–267
41. Deutsch, A., Fernandez, M.F., Florescu, D., Levy, A., Maier, D., Suciu, D.: Querying XML data. Bull. of the IEEE Computer Society Technical Committee on Data Engineering **22** (1999) 10–18

# Containment of Conjunctive Queries with Safe Negation

Fang Wei and Georg Lausen

University of Freiburg, Institute for Computer Science, Germany
{fwei,lausen}@informatik.uni-freiburg.de

**Abstract.** We consider the problem of query containment for conjunctive queries with safe negated subgoals ($CQ^{\neg}$s). We propose a new method for the containment test of $CQ^{\neg}$s. Comparing to the previous known approach, which always requires an exponential number of canonical databases to be verified to prove that $Q_1 \sqsubseteq Q_2$, the algorithm proposed in this paper exploits the containment mappings of their positive counterparts, and terminates once the specified test succeeds. We show that in the worst case, the algorithm has the same performance as the one proposed in previous work. We also extend our algorithm to unions of $CQ^{\neg}$s in a natural way. Due to the close relation between query containment and answering queries using views, we give some notes on considering answering queries using views when both queries and views have safe negated subgoals.

## 1 Introduction

This paper considers the problem of query containment of conjunctive queries ($CQ$s) with safe negated subgoals $CQ^{\neg}$s. The query containment problem is to check whether the answer set of one query is always a subset of another query for all databases. Algorithms for query containment are of interest in several contexts in the database area.

Recently, there is a renewed interest in containment checking of conjunctive queries. The main motivation lies in its tight relation to the problem of *answering queries using views* [9,1], which arises as the central problem in data integration and data warehousing (see [14] for a survey). Furthermore, query containment has also been used for checking integrity constraints [6], and for deciding query independence of updates [10].

Based on the NP-completeness result proposed by Chandra and Merlin [2], many researchers have been working on extensions of the containment question. Containment of $CQ$s with inequalities is discussed in [8,15]. Containment of unions of $CQ$s is treated in [12], containment of $CQ$s with negated subgoals in [10,14], containment over complex objects in [11], and over semi-structured data with regular expressions in [5].

The containment problem for conjunctive queries with safe negated subgoals has drawn considerably less attention in the past. In [10] uniform containment is discussed, which is a sufficient, however not necessary condition for containment.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 346–360, 2003.

In [14] it is argued that the complexity of the containment test is $\Pi_2^p$-complete. An algorithm based on the approach of *canonical databases* was sketched which tests an exponential number of canonical databases. The following example explains the approach:

*Example 1.* Consider the following queries $Q_1$ and $Q_2$:

$$Q_1 : \mathtt{q}(X, Z) \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z), \neg\mathtt{a}(X, Z).$$
$$Q_2 : \mathtt{q}(A, C) \text{ :- } \mathtt{a}(A, B), \mathtt{a}(B, C), \mathtt{a}(B, D), \neg\mathtt{a}(A, D).$$

In order to show that $Q_1 \sqsubseteq Q_2$, the approach in [14] considers all five partitions of $\{X, Y, Z\}$ in Table 1: all variables in one set of a certain partition are replaced by the same constant. From each partition, a canonical database, built out of the positive subgoals, is generated according to the predicates in the body of $Q_1$. At first, $Q_1$ has to be applied to the canonical database $D$ from each partition, and if the answer set is not empty, then the same answer set has to be obtained from $Q_2(D)$. Next, for each canonical database $D$ which results in a nonempty answer, we have to extend it with "other tuples that are formed from the same symbols as those in $D$". In fact, for each specific predicate, let $k$ be the number of arguments of the predicate, $n$ be the number of symbols in the canonical database, and $r$ be the number of subgoals of $Q_1$ (both positive and negative), there will be $2^{(n^k - r)}$ sets of tuples which have to be checked. Taking the partition $\{X\}\{Y\}\{Z\}$, we need to consider 6 other tuples: $\{\mathtt{a}(0,0), \mathtt{a}(1,0), \mathtt{a}(1,1), \mathtt{a}(2,0), \mathtt{a}(2,1), \mathtt{a}(2,2)\}$. At the end, one has to check $2^6$ canonical databases, and if for each database $D'$, $Q_2(D')$ yields the same answer as $Q_1$, it can then be concluded that $Q_1 \sqsubseteq Q_2$, which is true in this example. □

**Table 1.** The five canonical databases and their answers to $Q_1$ and $Q_2$

| Partition | Canonical Databases | Answers |
|---|---|---|
| $\{X\}\{Y\}\{Z\}$ | $\{\mathtt{a}(0,1), \mathtt{a}(1,2)\}$ | $Q_1 : \mathtt{q}(0,2); Q_2 : \mathtt{q}(0,2)$ |
| $\{X,Y\}\{Z\}$ | $\{\mathtt{a}(0,0), \mathtt{a}(0,1)\}$ | $Q_1 :$ `false` |
| $\{X\}\{Y,Z\}$ | $\{\mathtt{a}(0,1), \mathtt{a}(1,1)\}$ | $Q_1 :$ `false` |
| $\{X,Z\}\{Y\}$ | $\{\mathtt{a}(0,1), \mathtt{a}(1,0)\}$ | $Q_1 : \mathtt{q}(0,0); Q_2 : \mathtt{q}(0,0)$ |
| $\{X,Y,Z\}$ | $\{\mathtt{a}(0,0)\}$ | $Q_1 :$ `false` |

*Example 2.* Consider the following queries $Q_1$ and $Q_2$:

$$Q_1 : \mathtt{q}(X, Z) \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z), \neg\mathtt{a}(X, Z).$$
$$Q_2 : \mathtt{q}(A, C) \text{ :- } \mathtt{a}(A, B), \mathtt{a}(B, C), \neg\mathtt{b}(C, C).$$

This example differs from Example 1 by the negated subgoal. The application of $Q_2$ to the canonical databases shown in Table 1 yields the same answer as $Q_1$.

Similar to the above example, extra tuples have to be added into the canonical database. Taking the partition $\{X\}\{Y\}\{Z\}$, we have 15 other tuples (9 tuples with $\mathtt{b}(0,0),\ldots,\mathtt{b}(2,2)$ and 6 tuples as in Example 1), such that $2^{15}$ canonical databases have to be verified. Since the database $D = \{\mathtt{a}(0,1), \mathtt{a}(1,2), \mathtt{b}(2,2)\}$ is a counter-example such that $Q_2(D)$ does not generate the same answer as $Q_1(D)$, the test terminates with the result $Q_1 \not\sqsubseteq Q_2$. □

In this paper, we propose a new method to solve the general query containment problem for conjunctive queries with safe negated subgoals ($CQ^{\neg}$s). Given two $CQ^{\neg}$s $Q_1$ and $Q_2$, and their positive counterparts ${Q_1}^+$ and ${Q_2}^+$ (definitions in Section 3.1), we show that there are two factors deciding the complexity of the problem $Q_1 \sqsubseteq Q_2$:

- ${Q_1}^+ \sqsubseteq {Q_2}^+$? This is a necessary condition.
- the number of containment mappings from ${Q_2}^+$ to ${Q_1}^+$.

Comparing to the algorithm described in [14], which requires always an exponential number of canonical database to be tested to prove the result $Q_1 \sqsubseteq Q_2$, the algorithm proposed in this paper exploits the containment mappings from $Q_2^+$ to $Q_1^+$, and terminates when the specified tests succeed. We show that in the worst case, the algorithm has the same performance as the one proposed in [14]. Our algorithm also extends naturally to unions of $CQ^{\neg}$s. Due to the close relation between query containment and answering queries using views, we give some notes on considering answering queries using views when both queries and views have safe negated subgoals allowed.

The rest of the paper is organized as follows: in Section 2 we recall the definition of a $CQ^{\neg}$ and containment of both $CQ$s and $CQ^{\neg}$s. In Section 3, we first prove two necessary conditions for the containment test of $CQ^{\neg}$s, which is then followed by the main theorem of the paper. The proof of correctness and completeness is given as well. In Section 4 the theorem and the algorithm based on it are naturally extended to the containment test of unions of $CQ^{\neg}$s. In Section 5 we discuss the issues of answering queries using views when both queries and views have negated subgoals allowed. Finally the conclusions and future work are presented.

## 2 Preliminaries

### 2.1 Query Containment

A *conjunctive query with negation* ($CQ^{\neg}$) extends a *conjunctive query* $CQ$ by allowing negated subgoals in the body. It has the following form:

$$h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$

where $h, p_1, \ldots, p_n, s_1, \ldots, s_m$ are predicates whose arguments are variables or constants, $h(\bar{X})$ is the *head*, $p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$ are the positive subgoals, and

$s_1(\bar{Y_1}), \ldots, s_m(\bar{Y_m})$ are the negated subgoals. We assume that, firstly, the variables occurring in the head also occur in the body; secondly, all the variables occurring in the negated subgoals also occur in positive ones, which is also called the *safeness* condition for $CQ^\neg$. The examples in this paper are *safe* $CQ^\neg$s if not mentioned otherwise.

A $CQ^\neg$ is applied to a set of finite database relations by considering all possible substitutions of values for the variables in the body. If a substitution makes all the positive subgoals true and all the negated subgoals false (i.e. they do not exist in the database), then the same substitution, applied to the head, composes one answer of the conjunctive query. The set of all answers to a query $Q$ with respect to a certain database $D$ is denoted by $Q(D)$.

Unlike CQs with only positive subgoals, which are always satisfiable, $CQ^\neg$s might be unsatisfiable.

**Proposition 1.** *A $CQ^\neg$ is unsatisfiable if and only if there exist $p_i(\bar{X}_i)(1 \leq i \leq n)$ and $s_j(\bar{Y}_j)(1 \leq j \leq m)$ such that $p_i = s_j$ and $\bar{X}_i = \bar{Y}_j$.* □

From now on, we only refer to satisfiable $CQ^\neg$s, if not otherwise mentioned.

The containment of $CQ^\neg$s is defined in the same manner as for positive ones: a $CQ^\neg$ $Q_1$ is contained in another one $Q_2$, denoted as $Q_1 \sqsubseteq Q_2$, if for all databases $D$, $Q_1(D) \subseteq Q_2(D)$. Two $CQ^\neg$s are equivalent if and only if they are contained in each other.

### 2.2 The Containment Checking Algorithm for *CQ*s

An algorithm for checking the containment of $CQ$s was proposed in [2].

**Lemma 1 (Containment of $CQ$s [2]).** *Consider two $CQ$s $Q_1$ and $Q_2$:*

$$Q_1 : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}).$$
$$Q_2 : h(\bar{U}) \text{ :- } q_1(\bar{U_1}), \ldots, q_l(\bar{U_l}).$$

*Then $Q_1 \sqsubseteq Q_2$ if and only if there exists a containment mapping $\rho$ from the variables of subgoals in $Q_2$ to those in $Q_1$, such that $\{\rho(q_1(\bar{U_1})), \ldots, \rho(q_l(\bar{U_l}))\} \subseteq \{p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})\}$, and $\rho(h(\bar{U})) = h(\bar{X})$.* □

When the heads of both $Q_1$ and $Q_2$ do not contain variables, they are called boolean queries. It is obvious that boolean queries are the generalized forms of normal queries. The containment problem for $CQ$s is shown to be NP-complete [2].

*Example 3.* Consider the queries $Q_1$ and $Q_2$: the bodies of the queries are composed of the positive subgoals from Example 1.

$$Q_1 : \mathtt{q}(X, Z) \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z).$$
$$Q_2 : \mathtt{q}(A, C) \text{ :- } \mathtt{a}(A, B), \mathtt{a}(B, C), \mathtt{a}(B, D).$$

There is one and *only one* containment mapping from $Q_2$ to $Q_1$:
$\{A \rightarrow X, B \rightarrow Y, C \rightarrow Z, D \rightarrow Z\}$. □

## 3 Query Containment for $CQ^{\neg}$s

In this section we discuss the containment checking for $CQ^{\neg}$s. In the next subsection we introduce some necessary conditions.

### 3.1 Some Necessary Conditions

**Definition 1 (Super-Positive SP $Q^+$).** *Given a $CQ^{\neg}$ $Q$ as follows:*

$$Q : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$

*The SP of $Q$, denoted as $Q^+$ is: $h(\bar{X})$ :- $p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$.* □

**Lemma 2.** *Given a $CQ^{\neg}$ $Q$ with negated subgoals and its SP $Q^+$, $Q \sqsubseteq Q^+$ holds.* □

**Proposition 2.** *Let $Q_1$ and $Q_2$ be two $CQ^{\neg}$s, let $Q_1^+$ and $Q_2^+$ be their SP respectively. $Q_1 \sqsubseteq Q_2$ only if $Q_1^+ \sqsubseteq Q_2^+$.*

*Proof.* Assume $Q_1 \sqsubseteq Q_2$ and a tuple $t \in Q_1^+(D)$ where $D$ is any canonical database (i.e. each variable is assigned to a unique constant) of $Q_1^+$. We show that $t \in Q_2^+(D)$: Let $\rho$ be the substitution from variables of $Q_1$ to distinct constants in $D$. Let $s_i(\bar{Y_i})(1 \leq i \leq m)$ be any negated subgoal in $Q_1$. Since $Q_1$ is satisfiable, therefore we obtain that $\rho(s_i(\bar{Y_i})) \notin D$. Consequently, $t \in Q_1(D)$ and $t \in Q_2(D)$ are obtained. Following Lemma 2 it is obvious that $t \in Q_2^+(D)$.

Proposition 2 provides a necessary but not sufficient condition for query containment of $CQ^{\neg}$s. Next we give a theorem, stating a condition for $Q_1 \not\sqsubseteq Q_2$.

**Theorem 1.** *Let $Q_1$ and $Q_2$ be two $CQ^{\neg}$s. Assume $Q_1^+ \sqsubseteq Q_2^+$, and let $\rho_1, \ldots, \rho_r$ be all containment mappings from $Q_2^+$ to $Q_1^+$, such that $Q_1^+ \sqsubseteq Q_2^+$. $Q_1$ and $Q_2$ are given as follows:*

$$Q_1 : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$
$$Q_2 : h(\bar{U}) \text{ :- } q_1(\bar{U_1}), \ldots, q_l(\bar{U_l}), \neg a_1(\bar{W_1}), \ldots, \neg a_k(\bar{W_k}).$$

*If for each $\rho_i(1 \leq i \leq r)$, there exists at least one $j(1 \leq j \leq k)$, such that $\rho_i(a_j(\bar{W_j})) \in \{p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})\}$, then $Q_1 \not\sqsubseteq Q_2$.*

*Proof.* A canonical database $D$ could be constructed as follows: freeze the positive subgoals of $Q_1$ and replace each variable in $Q_1$ with a distinct constant. We call this substitution $\sigma$. Let $t$ be any tuple such that $t \in Q_1(D)$, we have to show that $t \notin Q_2(D)$: that is, for each substitution $\theta$ which makes $t \in Q_2^+(D)$ true, there is at least one negated subgoal $a_j(\bar{W_j})$, where $1 \leq j \leq k$, such that $\theta(a_j(\bar{W_j})) \in D$.

Since $\rho_1, \ldots, \rho_r$ are *all* the containment mappings from $Q_2^+$ to $Q_1^+$, it is true that $\theta \in \{\rho_1 \circ \sigma, \ldots, \rho_r \circ \sigma\}$. [1] Assume $\theta = \rho_i \circ \sigma$ $(1 \leq i \leq r)$. Since for each $\rho_i$, there exists a $j(1 \leq j \leq k)$, such that $\rho_i(a_j(\bar{W_j})) \in \{p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})\}$, thus we have $\rho_i \circ \sigma(a_j(\bar{W_j})) \in \{\sigma(p_1(\bar{X_1})), \ldots, \sigma(p_n(\bar{X_n}))\}$ As a result, $\theta(a_j(\bar{W_j})) \in D$ is obtained.

[1] $\rho \circ \sigma$ denotes the composition of substitutions $\rho$ and $\sigma$.

### 3.2 Containment of $CQ^{\neg}$s

The following theorem states a necessary and sufficient condition for the containment checking of $CQ^{\neg}$s, which is one of the main contributions of this paper.

**Theorem 2.** *Let $Q_1$ and $Q_2$ be two $CQ^{\neg}$s as follows:*

$$Q_1 : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$
$$Q_2 : h(\bar{U}) \text{ :- } q_1(\bar{U_1}), \ldots, q_l(\bar{U_l}), \neg a_1(\bar{W_1}), \ldots, \neg a_k(\bar{W_k}).$$

*Then $Q_1 \sqsubseteq Q_2$ if and only if*

1. *there is a containment mapping $\rho$ from $Q_2^+$ to $Q_1^+$ such that $Q_1^+ \sqsubseteq Q_2^+$, and*
2. *for each $j(1 \leq j \leq k)$, $Q' \sqsubseteq Q_2$ holds, where $Q'$ is as follows:*

$$Q' : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}), \rho(a_j(\bar{W_j})).$$

*Proof.*

- $\Leftarrow$: Let $D$ be any database and $t$ the tuple such that $t \in Q_1(D)$, we have to prove that $t \in Q_2(D)$.
  Since $t \in Q_1(D)$, we have immediately $t \in Q_1^+(D)$ and $t \in Q_2^+(D)$. Let $\sigma$ be the substitution from the variables in $Q_1^+$ to the constants in $D$ such that $t \in Q_1^+(D)$. Let $\theta = \rho \circ \sigma$. It is apparent that $\{\theta(q_1(\bar{U_1})), \ldots, \theta(q_l(\bar{U_l}))\} \subseteq D$. If for each $j(1 \leq j \leq k)$, $\theta(a_j(\bar{W_j}))) \notin D$, then the result is straightforward. Otherwise, if there is any $j(1 \leq j \leq k)$, such that $\theta(a_j(\bar{W_j}))) \in D$, then we have $\rho \circ \sigma(a_j(\bar{W_j}))) \in D$. it can be concluded that $t \in Q'(D)$ where $Q'$ is

  $$Q' : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}), \rho(a_j(\bar{W_j})).$$

  From the assumption that $Q' \sqsubseteq Q_2$ in the above theorem, $t \in Q_2(D)$ can then be obtained.
- $\Rightarrow$: The proof is via deriving a contradiction.
  1. If $Q_1^+ \not\sqsubseteq Q_2^+$, then from Proposition 2, $Q_1 \not\sqsubseteq Q_2$ can be obtained immediately.
  2. Otherwise, if for each containment mapping $\rho$ from ${Q_2}^+$ to ${Q_1}^+$, such that ${Q_1}^+ \sqsubseteq {Q_2}^+$, there is at least one $Q'$ as given in the above theorem, such that $Q' \not\sqsubseteq Q_2$, then there exists at least one database $D$, such that $t \in Q'(D)$, but $t \notin Q_2(D)$. Since $Q'$ has only one more positive subgoal than $Q_1$, it is obvious that $t \in Q_1(D)$. This leads to the result that $Q_1 \not\sqsubseteq Q_2$.

Theorem 2 involves a recursive containment test. In each round, the containment $Q' \sqsubseteq Q_2$ (the definition of $Q'$ see the above theorem) has to be verified. This might lead to one of the two results: (1) for each $Q'$, there is $Q' \sqsubseteq Q_2$ – either $Q'$ is unsatisfiable, or via recursive containment test –, then the test terminates with the result $Q_1 \sqsubseteq Q_2$; (2) there exists a $Q'$, such that $Q' \not\sqsubseteq Q_2$. This can be verified according to Theorem 1. In this case, the result of $Q_1 \not\sqsubseteq Q_2$ can be obtained. The following example illustrates the algorithm. Note that we intentionally omit the variables in the head in order to *generate* more containment mappings from $Q_2^+$ to $Q_1^+$. It is not difficult to understand that the fewer containment mappings are there from $Q_2^+$ to $Q_1^+$, the simpler the test will be.

*Example 4.* Given the queries $Q_1$ and $Q_2$:

$$Q_1 : h \text{ :- } \mathtt{a}(X,Y), \mathtt{a}(Y,Z), \neg\mathtt{a}(X,Z).$$
$$Q_2 : h \text{ :- } \mathtt{a}(A,B), \mathtt{a}(C,D), \neg\mathtt{a}(B,C)$$

There are four containment mappings from $Q_2^+$ to $Q_1^+$, one of which is $\rho_1 = \{A{\rightarrow}Y, B{\rightarrow}Z, C{\rightarrow}X, D{\rightarrow}Y\}$. Now a new conjunctive query is generated as follows:

$$Q' : h \text{ :- } \mathtt{a}(X,Y), \mathtt{a}(Y,Z), \mathtt{a}(Z,X), \neg\mathtt{a}(X,Z).$$

Note that the subgoal $\mathtt{a}(Z,X)$ is generated from $\rho_1(\mathtt{a}(B,C))$. One of the containment mappings from $Q_2$ to $Q'$ is $\rho_2 = \{A{\rightarrow}Z, B{\rightarrow}X, C{\rightarrow}Z, D{\rightarrow}X\}$. Since the newly generated subgoal $\rho_2(\mathtt{a}(B,C))$ is $\mathtt{a}(X,Z)$, this leads to a successful unsatisfiability test of $Q''$.

$$Q'' : h \text{ :- } \mathtt{a}(X,Y), \mathtt{a}(Y,Z), \mathtt{a}(Z,X), \mathtt{a}(X,Z), \neg\mathtt{a}(X,Z).$$

it can then be concluded that $Q' \sqsubseteq Q_2$. In the sequel we have $Q_1 \sqsubseteq Q_2$. □

The detailed algorithm is given in the Appendix. The idea behind the algorithm can be informally stated as follows: we start with the positive subgoals of $Q_1$ as root. Let $r$ be the number of all containment mappings from $Q_2^+$ to $Q_1^+$, such that $Q_1^+ \sqsubseteq Q_2^+$. $r$ branches are generated from the root, with sets of mapped negated subgoals as nodes (cf. Figure 1(a)). Next, each node might be marked as *Contained*, if it is identical to one of the negated subgoals of $Q_1$, or as *Terminal*, if it is identical to one of the positive subgoals of $Q_1$. If there exists one branch such that each node is marked as *Contained*, then the program terminates with the result $Q_1 \sqsubseteq Q_2$. Otherwise, if at least one node of each branch is marked as *Terminal*, then the program terminates too, however with the result $Q_1 \not\sqsubseteq Q_2$. If none of these conditions is met, the program continues with expansion of the non-terminal nodes, that is, the nodes mark with *Terminal* will not be expanded any more.

It can be shown that the algorithm terminates. The reasons are: (1) the expansion does not generate new variables; (2) the number of variables in $Q_1$ is finite.

The next example shows how the algorithm terminates when the complement problem $Q_1 \not\sqsubseteq Q_2$ is solved.

*Example 5.* Given the queries $Q_1$ and $Q_2$:

$$Q_1 : h \text{ :- } \mathtt{a}(X,Y), \mathtt{a}(Y,Z), \neg\mathtt{a}(X,Z).$$
$$Q_2 : h \text{ :- } \mathtt{a}(A,B), \mathtt{a}(C,D), \neg\mathtt{a}(A,D), \neg\mathtt{a}(B,C)$$

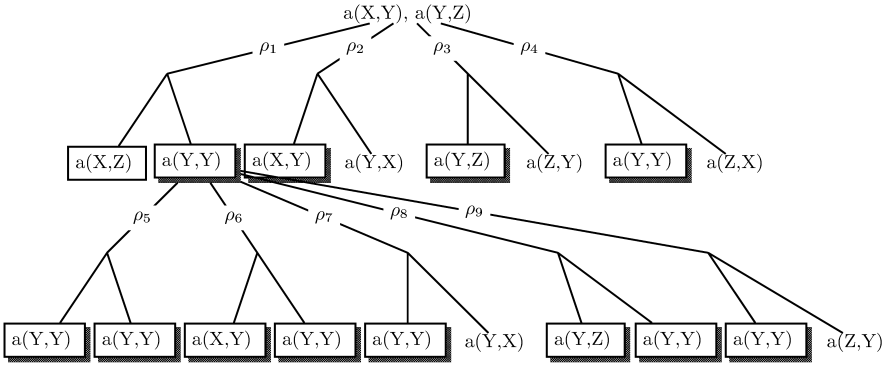
In Figure 1, it is shown that there are four containment mappings from $Q_2^+$ to $Q_1^+$: $\rho_1, \ldots, \rho_4$. Each mapping contains two sub-trees since there are two negated subgoals in $Q_2$. The branches $\rho_2$ and $\rho_3$ are marked as *Terminal* because there is at least one *Terminal* node from each of the above branches (note that we denote a node *Terminal* with a shadowed box around it, and *Contained* with a normal

box). The node $\mathtt{a}(X, Z)$ from branch $\rho_1$ is marked as *Contained*, because it is the same as the negated subgoal in $Q_1$. (Note that in Figure 1 the node $\mathtt{a}(Y, Y)$ of branch $\rho_1$ is marked as *Terminal*, but it is the result of the next round. Up to now, it has not been marked. The same holds for the nodes of branch $\rho_4$.)

Next the non-terminal node $\mathtt{a}(Y, Y)$ is expanded. Five new containment mappings are generated as $\rho_5, \ldots, \rho_9$. Since all the branches are *Terminal*, and $\mathtt{a}(Y, Y)$ is also a sub-node of $\rho_4$, it can be concluded that the expanded query

$$Q' : h \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z), \mathtt{a}(Y, Y), \neg\mathtt{a}(X, Z).$$

is not contained in $Q_2$. Because all the containment mappings from $Q_2^+$ to $Q'^+$ have been verified. As a result, $Q_1 \not\sqsubseteq Q_2$ is obtained. □




(a) The generated tree

| | | | |
|---|---|---|---|
| $\rho_1$ | $\{A \to X, B \to Y, C \to Y, D \to Z\}$ | $\rho_2$ | $\{A \to X, B \to Y, C \to X, D \to Y\}$ |
| $\rho_3$ | $\{A \to Y, B \to Z, C \to Y, D \to Z\}$ | $\rho_4$ | $\{A \to Y, B \to Z, C \to X, D \to Y\}$ |
| $\rho_5$ | $\{A \to Y, B \to Y, C \to Y, D \to Y\}$ | $\rho_6$ | $\{A \to X, B \to Y, C \to Y, D \to Y\}$ |
| $\rho_7$ | $\{A \to Y, B \to Y, C \to X, D \to Y\}$ | $\rho_8$ | $\{A \to Y, B \to Y, C \to Y, D \to Z\}$ |
| $\rho_9$ | $\{A \to Y, B \to Z, C \to Y, D \to Y\}$ | | |

(b) The containment mappings

**Fig. 1.** The graphic illustration of Example 5.

*Comparison of the Algorithms.* We notice several interesting similarities and differences between our algorithm and the one in [14]: The partitioning of variables is similar to the step of checking of ${Q_1}^+ \sqsubseteq {Q_2}^+$. It can be proven that if the containment checking ${Q_1}^+ \sqsubseteq {Q_2}^+$ is successful, there exists at least one partition of variables, namely the partition with a distinct constant for each variable, such that when applied to the canonical database built from this partition, $Q_2$ yields the same answer set as $Q_1$.

The next step of the algorithm in [14] is to check an exponential number of canonical databases, as described in the Introduction. In contrast, our algorithm continues with the containment mappings of their positive counterparts and executes the specified test, which takes linear time in the size of $Q_1$. If the test is not successful, the query is extended with one more positive subgoal (but without new variables) and the next containment test continues. It is important to emphasize that in the worst case, the expanded tree generates all the nodes composed of variant combinations of the variables in $Q_1$, which coincides with the method in [14].

However, in the following examples we show that our algorithm terminates *definitely* earlier in the some cases:

(1) It turns out that the result of the containment test is $Q_1 \sqsubseteq Q_2$. As explained in the Introduction, the algorithm in [14] terminates if one canonical database as counter-example is found, but in order to obtain the result of $Q_1 \sqsubseteq Q_2$, *all* canonical databases have to be verified. In contrast, our algorithm terminates with the result $Q_1 \sqsubseteq Q_2$ if the specified test is successful.

*Example 6.* Given the queries $Q_1$ and $Q_2$ as in Example 1. There is a containment mapping $\rho$ from $Q_2^+$ to $Q_1^+$ as in Example 3. There is only one $Q'$ to be checked:

$$Q' : \mathtt{q}(X, Z) \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z), \neg\mathtt{a}(X, Z), \mathtt{a}(X, Z).$$

It is apparent that $Q'$ is unsatisfiable, thus we have proven that $Q_1 \sqsubseteq Q_2$. □

(2) To test $Q_1 \sqsubseteq Q_2$, if none of the predicates from negated subgoals in $Q_2$ appears in positive subgoals, the test terminates after the first round of containment checking of $Q' \sqsubseteq Q_2$. It can be explained as follows: due to the assumption, the containment mappings from $Q_2$ to the newly generated query $Q'$ (see Theorem 2) are the same as the ones from $Q_2$ to $Q_1$. Since no *new* containment mapping is generated, the algorithm reaches some "fix-point", so that no new branches of the expanded tree will be generated.

*Example 7.* Given the queries $Q_1$ and $Q_2$ as in Example 2. There is a containment mapping $\rho_1$ (cf. Figure 2). A new node $\rho_1(\mathtt{b}(C, C)) = \mathtt{b}(Z, Z)$ is then generated. The new query $Q'$ is the following:

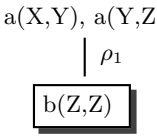
$$Q' : \mathtt{q}(X, Z) \text{ :- } \mathtt{a}(X, Y), \mathtt{a}(Y, Z), \neg\mathtt{a}(X, Z), \mathtt{b}(Z, Z).$$

Since $\rho_1$ is the *only* containment mapping from $Q_2$ to $Q'$, we mark the node $\mathtt{b}(Z, Z)$ as *Terminal*. Following Theorem 1, $Q' \not\sqsubseteq Q_2$ can be obtained, which is followed by the result $Q_1 \not\sqsubseteq Q_2$. □

At last, it should be mentioned that the algorithm in [14] can deal with unsafe negations, while ours cannot.

## 4 Containment of Unions of $CQ^{\neg}$s

In this section we consider the containment problem of unions of $CQ^{\neg}$s. First we present some basic notations.




(a) The generated tree

| $\rho_1$ | $\{A \to X, B \to Y, C \to Z\}$ |
|---|---|

(b) The containment mapping

**Fig. 2.** The graphic illustration of Example 7.

**Definition 2 (Union of $CQ$s [13]).** *Let $\mathcal{Q} = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ$s, in which $Q_1, \ldots, Q_n$ have a common head predicate.*

- *Given any database $D$, $\mathcal{Q}(D) = Q_1(D) \cup \ldots \cup Q_n(D)$.*
- *Let $Q$ be a $CQ$ and $\mathcal{Q}$ be a union of $CQ$s. $Q \sqsubseteq \mathcal{Q}$ if $Q(D) \subseteq \mathcal{Q}(D)$ for any given database $D$.* □

**Theorem 3 (Containment of unions of $CQ$s [12]).** *Let $Q$ be a $CQ$ and $\mathcal{Q} = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ$s, then $Q \sqsubseteq \mathcal{Q}$ if and only if there is a $Q_i (1 \leq i \leq n)$ such that $Q \sqsubseteq Q_i$.* □

However, when negation is allowed in $CQ$s, the above theorem will not hold any more.

*Example 8.* Consider the following queries with the relations `man`, `father`, and `husband`:

$$\begin{aligned} Q: \ & q(X) \text{ :- } \texttt{man}(X), \neg\texttt{father}(X). \\ Q_1: & q(Y) \text{ :- } \texttt{man}(Y), \neg\texttt{husband}(Y). \\ Q_2: & q(Z) \text{ :- } \texttt{man}(Z), \texttt{husband}(Z), \neg\texttt{father}(Z). \end{aligned}$$

It is apparent that neither $Q \sqsubseteq Q_1$, nor $Q \sqsubseteq Q_2$, but $Q \sqsubseteq Q_1 \cup Q_2$. □

The containment of unions of $CQ^\neg$s is defined in the similar way as in Definition 2. The following notations have to be additionally presented.

**Definition 3.** *Let $\mathcal{Q} = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ^\neg$s. $\mathcal{Q}^+ = Q_1^+ \cup \ldots \cup Q_n^+$.* □

**Lemma 3.** *Let $\mathcal{Q} = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ^\neg$s. $\mathcal{Q} \sqsubseteq \mathcal{Q}^+$ holds.* □

**Lemma 4.** *Given a $CQ^\neg$ $Q$ and a union of $CQ^\neg$s $\mathcal{Q} = Q_1 \cup \ldots \cup Q_n$. If $Q \sqsubseteq \mathcal{Q}$, then there exists at least one $Q_i \in \mathcal{Q} (1 \leq i \leq n)$, such that $Q^+ \sqsubseteq Q_i^+$.*

*Proof. (sketch)* Using the same method as in Proposition 2, we have $Q^+ \sqsubseteq Q_1^+ \cup \ldots \cup Q_n^+$. Following [12] the result can be obtained.

**Theorem 4.** *Given a $CQ^{\neg}$ $Q$ and a union of $CQ^{\neg}$s $\mathcal{Q} = Q_1 \cup Q_2 \cup \ldots \cup Q_v$. Assume $Q^+ \sqsubseteq \mathcal{Q}^+$, and let $\rho_{i,1}, \ldots, \rho_{i,r_i}$ (for $i = 1, \ldots, v$) be all the containment mappings from $\mathcal{Q}^+$ to $Q^+$, such that $Q^+ \sqsubseteq \mathcal{Q}^+$. $Q$ and $Q_1, \ldots, Q_v$ are given as follows:*

$$Q: \quad h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$
$$Q_i (1 \leq i \leq v): h(\bar{U}) \text{ :- } q_{i,1}(\bar{U_{i,1}}), \ldots, q_{i,l_i}(\bar{U_{i,l_i}}), \neg a_{i,1}(\bar{W_{i,1}}), \ldots, \neg a_{i,k_i}(\bar{W_{i,k_i}}).$$

*If for each $\rho_{i,j} (1 \leq i \leq v, 1 \leq j \leq r_i)$, there exists at least a $a_{i,u}(\bar{W_{i,u}})$, where $1 \leq u \leq k_i$, such that $\rho_{i,j}(a_{i,u}(\bar{W_{i,u}})) \in \{p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})\}$, then $Q \not\sqsubseteq \mathcal{Q}$.*

*Proof. (sketch)* A canonical database $D$ can be built in the similar way as in Theorem 1. Then it can be proven that a tuple $t \in Q_1(D)$, but $t \notin Q_2(D)$.

The following theorem states a sound and complete condition.

**Theorem 5.** *Let $Q$ be a $CQ^{\neg}$ and $\mathcal{Q}$ be the union of $CQ^{\neg}$s $\mathcal{Q} = Q_1 \cup Q_2 \cup \ldots \cup Q_v$ as follows:*

$$Q: \quad h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$
$$Q_i (1 \leq i \leq v): h(\bar{U}) \text{ :- } q_{i,1}(\bar{U_{i,1}}), \ldots, q_{i,l_i}(\bar{U_{i,l_i}}), \neg a_{i,1}(\bar{W_{i,1}}), \ldots, \neg a_{i,k_i}(\bar{W_{i,k_i}}).$$

*Then $Q \sqsubseteq \mathcal{Q}$ if and only if*

- *there is a containment mapping $\rho$ from $Q_u^+$ to $Q$, where $1 \leq u \leq v$, such that $Q^+ \sqsubseteq \mathcal{Q}^+$, and*
- *for each $j (1 \leq j \leq k_u)$, $Q' \sqsubseteq \mathcal{Q}$ holds, where $Q'$ is as follows:*

$$Q' : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}), \rho(a_{u,j}(\bar{W_{u,j}}).$$

□

The algorithm behind the theorem can be adapted with little modification from that of containment checking for two $CQ^{\neg}$s. Next we explain how the containment checking of Example 8 is executed.

*Example 9.* Consider the queries $Q$, $Q_1$ and $Q_2$ in Example 8. Since $Q_1$ is the only query whose positive part of body contains that of $Q$, the containment mapping $\rho_1 = \{Y \rightarrow X\}$ is obtained. The newly generated $Q'$ has the following form:

$$Q' : q(X) \text{ :- } \texttt{man}(X), \neg \texttt{father}(X), \texttt{husband}(X).$$

Since it is easy to check that $Q' \sqsubseteq Q_2$, it can then be concluded that $Q \sqsubseteq Q_1 \cup Q_2$.
□

## 5 Some Notes on Answering Queries Using Views

The problem of answering queries using views has been intensively studied recently, especially in the area of data integration (see [7] for a survey). The problem can be described as follows: given a query $Q$ over a database schema, and a set of view definitions $\mathcal{V}$ over the same schema, how to find a rewriting $Q'$ of $Q$, using *only* the views. The notations of *equivalent rewriting* and *maximally-contained rewriting* can be found in [7].

Due to the close relation between the problem of query containment and answering queries using views, we are interested in giving the discussion of some new issues for answering queries using views, when safe negation is allowed in both queries and views. Since the *completeness* of any rewriting algorithm depends on the query language in which the rewritings are expressed, it has to be specified whether queries with unions and negations are allowed for expressing the rewriting. In the following example, we show that even without applying negation on the views, the *equivalent rewriting* can only be found if unions are allowed in the rewriting.

*Example 10.* Let $Q$ and $V_1$, $V_2$ be the conjunctive queries as follows:

$$\begin{array}{l} Q: \ q(X,Y) \ \text{:-} \ a(X,Y). \\ V_1 : v_1(X,Y) \text{:-} \ a(X,Y), b(Y). \\ V_2 : v_2(X,Y) \text{:-} \ a(X,Y), \neg b(Y). \end{array}$$

There is an equivalent rewriting of $Q$ using the union of $V_1$ and $V_2$. □

One important issue concerning answering queries using views is to decide whether a view is useful for the given query. It can be informally stated as the following [7]: a view can be useful for a query if the set of relations it mentions overlaps with that of the query, and it selects some of the attributes selected by the query. When a query with both positive and negative subgoals is considered, this condition remains true for the positive subgoals, but not for the negative subgoals any more. It can be shown in the following example that an *equivalent rewriting* can only be found by *negating* the views.

*Example 11.* Let $Q$ and $V_1$, $V_2$ be the conjunctive queries as follows:

$$\begin{array}{l} Q: \ q(X,Y) \ \text{:-} \ p_0(X,Y), \neg p_1(X,X). \\ V_1 : v_1(X,Y) \text{:-} \ p_0(X,Y). \\ V_2 : v_2(X,Z) \text{:-} \ p_0(X,Y), p_1(X,Z). \end{array}$$

It can be shown that the rewriting $Q'$ is an equivalent one:

$$Q' : q(X,Y) \text{:-} \ v_1(X,Y), \neg v_2(X,X).$$

□

One difficulty in choosing the views for the negated subgoals is to decide whether a view $V$ can be useful for the query $Q$ with its negated form. The next example differs from Example 11 only on $V_2$.

*Example 12.* Let $Q$ and $V_1$, $V_2$ be the conjunctive queries as follows:

$$\begin{aligned} Q &: q(X,Y) \ \text{:-}\ p_0(X,Y), \neg p_1(X,X). \\ V_1 &: v_1(X,Y) \text{:-}\ p_0(X,Y). \\ V_2 &: v_2(X,Z) \text{:-}\ p_1(X,Z), p_2(X,Y). \end{aligned}$$

The rewriting $Q'$ as follows is not even a *contained rewriting* for $Q$.

$$Q' : q(X,Y) \text{:-}\ v_1(X,Y), \neg v_2(X,X).$$

When $Q' \cup V_1 \cup V_2$ is applied to the database $D = \{p_0(0,1), p_1(0,0)\}$, tuple $q(0,1)$ is obtained. However it is apparent that this is not the correct answer for $Q$. □

Informally speaking, in order to be useful for the query $Q$ with its negated form, a view $V$ should not contain *foreign* predicates, which do not appear in $Q$ (cf. Example 12). Moreover, $V$ has to contain at least one positive subgoal whose negated form appears in $Q$, and the attributes of the subgoal should not be projected out.

*Related Work.* There is relatively less research concerning the problem of answering queries using views with safe negations appearing in the body of both queries and views. [4] provides an algorithm dealing with it. It is an extension of the *inverse rule* algorithm. However it is not clear whether the algorithm is complete w.r.t the problem of views based query answering.

## 6 Conclusion and Further Research

We have discussed the query containment problem for the conjunctive queries with safe negated subgoals. A new method for testing the containment of $CQ^{\neg}$s is given, comparing with the one in [14]. We have also shown that this algorithm can be naturally extended to the containment checking of unions of $CQ^{\neg}$s. At last, we have discussed the problem of answering queries using views when both queries and views have negated subgoals. Some motivating examples are given to show the cases which might not be encountered with pure conjunctive queries.

There are several interesting questions left for research. In [1] several categories of complexities for *answering queries using views* are given. However, with respect to negation, $Datalog^{\neg}$ is considered, and the setting of $CQ^{\neg}$ is not covered.

One interesting extension of our work is to consider the containment problem of CQs with *both* safe negation and arithmetic comparisons. To the best of our knowledge, no practical algorithm has been published to solve this problem. The complexity of this problem is still unknown as well.

**Acknowledgments.** We would like to thank Michael Benedikt for the valuable comments on this material.

## References


1. S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *ACM Symp. on Principles of Database Systems (PODS)*, 1998.
2. A. K. Chandra and P. M. Merlin. Optimal implementations of conjunctive queries in relational data bases. In *ACM Symp. on Theory of Computing (STOC)*, pages 77–90. 1977.
3. O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 778–784, 1997.
4. S. Flesca and S. Greco. Rewriting queries using views. In *TKDE*, 13(6), 2001.
5. D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *ACM Symp. on Principles of Database Systems (PODS)*. 1998.
6. A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *ACM Symp. on Principles of Database Systems (PODS)*. 1994.
7. A. Halevy. Answering queries using views: A survey. In *VLDB Journal*, 10:4, pp. 270–294, 2001.
8. A. Klug. On conjunctive queries containing inequalities. In *Journal of the ACM* 35:1, pp. 146–160, 1988.
9. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *ACM Symp. on Principles of Database Systems (PODS)*. 1995.
10. A. Levy and Y. Sagiv. Queries independent of updates. In *International Conference on Very Large Data Bases (VLDB)*, 1993.
11. A. Levy and D. Suciu. Deciding containment for queries with complex objects. In *ACM Symp. on Principles of Database Systems (PODS)*. ACM Press, 1997.
12. Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operations. *Journal of the ACM, (4)*, 27(4):633–655, 1980.
13. J. Ullman. *Principles of Database and KnowledgeBase Systems, Volume II.* 1989.
14. J. D. Ullman. Information integration using logical views. In *International Conference on Database Theory (ICDT)*, 1997.
15. X. Zhang and Z. Meral Özsoyoglu. On efficient reasoning with implication constraints. In *DOOD*, pages 236–252, 1993.


## APPENDIX

**Algorithm** Containment Checking$(Q_1, Q_2)$
**Inputs:** $Q_1$ and $Q_2$ are $CQ^\neg$s with the form as follows:

$$Q_1 : h(\bar{X}) \text{ :- } p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}), \neg s_1(\bar{Y_1}), \ldots, \neg s_m(\bar{Y_m}).$$
$$Q_2 : h(\bar{U}) \text{ :- } q_1(\bar{U_1}), \ldots, q_l(\bar{U_l}), \neg a_1(\bar{W_1}), \ldots, \neg a_k(\bar{W_k}).$$

**Begin**
1. Set $\{p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})\}$ as the root of a tree: $c_0$.
Let $\rho_1, \ldots, \rho_r$ be all the containment mappings from $Q_2^+$ to $Q_1^+$.
2. Generate $r$ nodes $c_1, \ldots, c_r$ as children of root, and each node $c_i (1 \leq i \leq r)$ has a subtree with $k$ children, in which each child $c_{i,j}$ with the form $\rho_i(a_j(\bar{W_j}))(1 \leq j \leq k)$.

3. Marking the nodes:
**For** i=1 to r **do**
  **For** j=1 to k **do**
    **If** $c_{i,j} \in \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\}$ **Then** mark $c_i$ as *Terminal*;
    **If** $c_{i,j} \in \{s_1(\bar{Y}_1), \ldots, s_m(ymbar)\}$ **Then** mark $c_{i,j}$ as *Contained*;
  **EndFor**
**EndFor**
4. Execute the containment checking:
**If** all nodes $c_1, \ldots, c_r$ are terminal nodes, **Then return** `Not-contained`;
**For** i=1 to r **do**
  **If** each $c_{i,j} (1 \leq j \leq k)$ is marked as *Contained*, **Then return** `Contained`;
**EndFor**
5. Continue expanding non-terminal nodes:
**For** i=1 to r **do**
  **If** $c_i$ is not *Terminal* **Then**
    **For** j = 1 to k **do**
      **If** $c_{i,j}$ is not *Contained* **Then**
        let $Q_1'$ be: $h$ :- $p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), c_{i,j}, \neg s_1(\bar{Y}_1), \ldots, \neg s_m(\bar{Y}_m)$.;
        let $\rho_{i,j,1}, \ldots, \rho_{i,j,b_{i,j}}$ be the new containment mappings from $Q_2$ to $Q_1'$;
        Generate $b_{i,j}$ nodes with children in the same way as in **Step 2**;
        Mark the nodes in the same way as **Step 3**;
      **EndIf**
    **EndFor**
  **EndIf**
**EndFor**
6. Execute the containment checking:
**For** i=1 to r **do**
  **If** each $c_{i,j} (1 \leq j \leq k)$ either is marked as *Contained*
  or has a child whose children are all marked *Contained*
  **Then return** `Contained`;
  **EndIf**
  from each non-terminal node $c_i$, choose one child $c_{i,j} (1 \leq j \leq k)$;
  add all these $c_{i,j}$ to the body of $Q_1$, and let the new query be $Q_1''$;
 **If** each branch w.r.t. $Q_1''$ is marked as *Terminal*, **Then return** `Not-contained`;
**EndFor**
Let $r$ be all expanded nodes; **Go to** Step 5;
**End**

# Probabilistic Interval XML

Edward Hung, Lise Getoor*, and V.S. Subrahmanian**

Dept. of Computer Science
University of Maryland, College Park, MD 20742
{ehung,getoor,vs}@cs.umd.edu

**Abstract.** Interest in XML databases has been growing over the last few years. In this paper, we study the problem of incorporating probabilistic information into XML databases. We propose the Probabilistic Interval XML (***PIXml*** for short) data model in this paper. Using this data model, users can express probabilistic information within XML markups. In addition, we provide two alternative formal model-theoretic semantics for ***PIXml*** data. The first semantics is a "global" semantics which is relatively intuitive, but is not directly amenable to computation. The second semantics is a "local" semantics which is more amenable to computation. We prove several results linking the two semantics together. To our knowledge, this is the first formal model theoretic semantics for probabilistic interval XML. We then provide an operational semantics that may be used to compute answers to queries and that is correct for a large class of probabilistic instances.

## 1 Introduction

Over the last few years, there has been considerable interest in XML databases. Our goal in this paper is twofold: first, to provide an extension of XML data model so that uncertain information can be stored, and second, to provide a formal model theory for such a stored XML database.

There are numerous applications (including financial, image processing, manufacturing and bioinformatics) for which a probabilistic XML data model is quite natural and for which a query language that supports probabilistic inferences provides important functionality. Probabilistic inferences support capabilities for predictive and 'what-if' types of analysis. For example, consider the use of a variety of predictive programs[3] for the stock market. Such programs usually return probabilistic information. If a company wanted to export this data into an XML format, they would need methods to store probabilistic data in XML. The financial marketplace is a hotbed of both predictive and XML activity (e.g. the FIX standard for financial data is XML based). The same applies for programs that predict expected energy usage and cost, expected failure rates for machine parts, and in general, to any predictive program. Another useful class of applications where there is a need for probabilistic XML data is in the case of image processing programs that process images (automatically) using image identification methods and store the results on the web. Such image processing algorithms often use statistical classifiers [11] and often yield uncertain data as output. If such information

* University of Maryland Institute for Advanced Computer Studies
** Institute for Advanced Computer Studies and Institute for Systems Research

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 361–377, 2003.

is to be stored in an XML database, then it would be very useful to have the ability to automatically query this uncertain information. A third application is in automated part diagnosis. A corporate manufacturing floor may use sensors to track what happens on the manufacturing floor. The results of the sensor readings may be automatically piped to a fault diagnosis program that may identify zero, one, or many possible faults with a variety of probabilities on the space of faults. When such analysis is stored in a database, there is a natural need for probabilities. In addition to these types of applications, the NSIR system for searching documents at the University of Michigan[19] returns documents based along with probabilities. Likewise, Nierman, et. al.[17] point out the use of probabilistic semistructured databases in protein chemistry.

These are just a few examples of the need for probabilistic XML data. In this paper, we will first develop the ***PIXml*** probabilistic interval data model which uses interval probabilities rather than point probabilities to represent uncertainty (a companion paper[10] describes an approach which uses only point probabilities, and gives an algebra and a query language along with experimental results). We will then provide two alternative formal semantics for ***PIXml*** . The first semantics is a declarative (model-theoretic) semantics. The second semantics is an operational semantics that can be used for computation. W3C's formal specification of XML considers an instance as an ordered rooted tree in which cycles can possibly appear[21]. Here, to guarantee coherent semantics for our model, we assume that an instance is an acyclic graph. We also provide an operational semantics that is provably correct for a queries over a large class of probabilistic instances called tree-structured instances.

## 2 Motivating Example

Consider a surveillance application where a battlefield is being monitored. Image processing methods are used to classify objects appearing in images. Some objects are classified as vehicle convoys or refugee groups. Vehicle convoys may be further classified into individual vehicles, which may be further classified into categories such as tanks, cars, armored personnel carriers. However, there may be uncertainty over the number of vehicles in the convoy as well as the categorization of a vehicle. For example, image processing methods often use Bayesian statistical models to capture uncertainty in their identification of image objects. Further uncertainty may arise because image processing methods may not explicitly extract the identity of the objects. For instance, even if we are certain that a given vehicle is a tank, we might not be able to classify it further into a T-72 tank or a T-80 tank. Semistructured data is a natural way to store such data because for a surveillance application of this kind, we have some idea of what the structure of data looks like (e.g. the general hierarchical structure alluded to above). However, the above example demonstrates the need of a semistructured model to store uncertain information in probabilistic environments.

## 3 Preliminaries

Here we provide some necessary definitions on which our ***PIXml*** model will be built.

### 3.1 Semistructured Databases

In this subsection, we quickly review the formal definition of a semistructured database and illustrate it through an example from the surveillance domain.

**Definition 1.** *A* **semistructured instance** $\mathcal{S}$ *over a set of objects* $\mathcal{O}$*, a set of labels* $\mathcal{L}$*, a set of types* $\mathcal{T}$ *is a* 5*-tuple* $\mathcal{S} = (V, E, \ell, \tau, \mathsf{val})$ *where:*

1. $G = (V, E, \ell)$ *is a rooted, directed graph where* $V \subseteq \mathcal{O}$*,* $E \subseteq V \times V$ *and* $\ell : E \rightarrow \mathcal{L}$*;*
2. $\tau$ *associates a type in* $\mathcal{T}$ *with each leaf object* $o$ *in* $G$*.*
3. $\mathsf{val}$ *associates a value in the domain* $\mathsf{dom}(\tau(o))$ *with each leaf object* $o$*.*

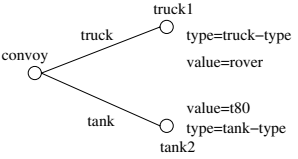



**Fig. 1.** A semistructured instance for the surveillance domain.

*Example 1.* Figure 1 shows a graph representing a part of the surveillance domain. The instance is defined over the set $\mathcal{O} = \{convoy, truck1, truck2, tank1, tank2\}$ of objects. The set of labels is $\mathcal{L} = \{truck, tank\}$. There are two types, *truck-type* and *tank-type*, with domains given by: $\mathsf{dom}(\textit{truck-type}) = \{mac, rover\}$ and $\mathsf{dom}(\textit{tank-type}) = \{t72, t80\}$. The graph shows that the instance consists of 3 objects: $convoy$, $truck1$ and $tank2$, connected by edges $(convoy, truck1)$ and $(convoy, tank2)$, labeled with $truck$ and $tank$ respectively. The types and values of the leaves are: $\tau(truck1) =$ *truck-type*, $\tau(tank2) =$ *tank-type*, $\mathsf{val}(truck1) = rover$ and $\mathsf{val}(tank2) = t80$.

### 3.2 Interval Probabilities

In this subsection, we quickly review definitions and give some important theorems for interval probabilities. Given an interval $I = [x, y]$ we will often use the notation $I.lb$ to denote $x$ and $I.ub$ to denote $y$.

An ***interval function*** $\iota$ w.r.t. a set $S$ associates, with each $s \in S$, a closed subinterval $[lb(s), ub(s)] \subseteq [0, 1]$. $\iota$ is called an ***interval probability function*** if $\sum_{s \in S} lb(s) \leq 1$ and $\sum_{s \in S} ub(s) \geq 1$. A ***probability distribution*** w.r.t. a set $S$ over an interval probability function $\iota$ is a mapping $\mathcal{P} : S \rightarrow [0, 1]$ where $\forall s \in S, lb(s) \leq \mathcal{P}(s) \leq ub(s)$ and $\Sigma_{s \in S} \mathcal{P}(s) = 1$.

**Lemma 1.** *For any set* $S$ *and any interval probability function* $\iota$ *w.r.t.* $S$*, there exists a probability distribution P(S) which is compatible with* $\iota$*.*

An interval probability function $\iota$ w.r.t. $S$ is ***tight*** iff for any interval probability function $\iota'$ w.r.t. $S$ such that every probability distribution $\mathcal{P}$ over $\iota$ is also a probability distribution over $\iota'$, $\iota(s).lb \geq \iota'(s).lb$ and $\iota(s).ub \leq \iota'(s).ub$ where $s \in S$. If every probability distribution $P$ over $\iota'$ is also a probability distribution over $\iota$, then we say that $\iota$ is the **tight equivalent** of $\iota'$. A ***tightening operator***, $\mathsf{tight}$, is a mapping from interval probability functions to interval probability functions such that $\mathsf{tight}(\iota)$ produces a tight equivalent of $\iota$. The following result (derived from Theorem 2 of [5]) tells us that we can always tighten an interval probability function.

**Theorem 1.** *Suppose $\iota, \iota'$ are interval probability functions over $S$ and $tight(\iota') = \iota$. Let $s \in S$. Then:*

$$\iota(s) = \left[ max\left(\iota'(s).lb, 1 - \sum_{s' \in S \wedge s' \neq s} \iota'(s').ub\right), min\left(\iota'(s).ub, 1 - \sum_{s' \in S \wedge s' \neq s} \iota'(s').lb\right)\right].$$

For example, we can use the above formula to check that the interval probability functions in Figure 2 are tight. Throughout the rest of this paper, unless explicitly specified otherwise, we will assume that all interval probability functions are tight.

## 4 The *PIXml* Data Model

Consider the case where there are two boolean state variables, $e_1$ and $e_2$, and thus four possible worlds depending on which combination of $e_1, e_2$ are true. In classical probabilistic logic ([8]), a statement such as "The probability of $e_1$ or $e_2$ occurring is $0.25$" can be viewed as saying that while we do not know with certainty the state of the world, we know that only three of the above four worlds can make $(e_1 \vee e_2)$ true. Furthermore, the sum of the probabilities of these three worlds must equal $0.25$ for the above statement to be true.

In probabilistic XML, we have uncertainty because we do not know which of various possible semistructured instances is "correct." Further, rather than defining a point probability for each instance, we will use interval probabilities to give bounds on the probabilities for structure. We use a structure called a probabilistic semistructured instance that determines many possible semistructured instances compatible with it. In this section, we will first define a probabilistic semistructured instance. The following section will describe its model theoretic semantics.

We begin with the definition of a weak instance. A weak instance describes the objects that can occur in a semistructured instance, the labels that can occur on the edges in an instance and constraints on the number of children an object might have. We will later define a probabilistic instance to be a weak instance with some probabilistic attributes.

**Definition 2.** *A **weak instance** $\mathcal{W}$ with respect to $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{T}$ is a 5-tuple $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ where:*

1. *$V \subseteq \mathcal{O}$.*
2. *For each object $o \in V$ and each label $l \in \mathcal{L}$, $\mathsf{lch}(o, l)$ specifies the set of objects that* **may** *be children of $o$ with label $l$. We assume that for each object $o$ and distinct labels $l_1, l_2$, $\mathsf{lch}(o, l_1) \cap \mathsf{lch}(o, l_2) = \emptyset$. (This condition says that two edges with different labels cannot lead to the same child).*

3. $\tau$ *associates a type in* $\mathcal{T}$ *with each leaf vertex.*
4. $\mathsf{val}$ *associates a value in* $\mathsf{dom}(\tau(o))$ *with each leaf object* $o$.
5. $\mathsf{card}$ *is mapping which constrains the number of children with a given label* $l$. $\mathsf{card}$ *associates with each object* $o \in V$ *and each label* $l \in \mathcal{L}$, *an integer-valued interval function,* $\mathsf{card}(o, l) = [min, max]$, *where* $min \geq 0$, *and* $max \geq min$. *We use* $\mathsf{card}(o, l).min$ *and* $\mathsf{card}(o, l).max$ *to refer to the lower and upper bounds respectively.*

A weak instance implicitly defines – for each object and each label – a set of potential sets of children.

**Definition 3.** *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$ *and* $l$ *is a label. A set* $\mathsf{c}$ *of objects in* $V$ *is a* ***potential*** $l$*-child set of* $o$ *w.r.t. the above weak instance iff:*

1. *If* $o' \in \mathsf{c}$ *then* $o' \in \mathsf{lch}(o, l)$ *and*
2. *The cardinality of* $\mathsf{c}$ *lies in the closed interval* $\mathsf{card}(o, l)$.

*We use the notation* $\mathsf{PL}(o, l)$ *to denote the set of all potential* $l$*-child sets of* $o$.

We are now in a position to define the potential children of an object $o$.

**Definition 4.** *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$. *A* **potential child set** *of* $o$ *is any set* $Q$ *of subsets of* $V$ *such that* $Q = \bigcup H$ *where* $H$ *is a hitting set*[1] *of* $\{\mathsf{PL}(o, l) \mid (\exists o')o' \in \mathsf{lch}(o, l)\}$. *We use* $\mathsf{potchildren}(o)$ *to denote the set of all potential child sets of* $o$ *w.r.t. a weak instance.*

Once a weak instance is fixed, $\mathsf{potchildren}(o)$ is well defined for each $o$. We will use this to define the *weak instance graph*.

**Definition 5.** *Given a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$, *the* **weak instance graph**, $\mathcal{G}_{\mathcal{W}} = (V, E)$, *is a graph over the same set of nodes* $V$, *and for each pair of nodes* $o$ *and* $o'$, *there is an edge from* $o$ *to* $o'$ *iff* $o' \in \mathsf{potchildren}(o)$.

As we will see later, in order to assure coherent semantics for our models, we will require our weak instance graph to be acyclic. We are now ready to define the important concept of a probabilistic instance.

**Definition 6.** *A* ***probabilistic instance*** $\mathcal{I}$ *is a* 6*-tuple* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ *where:*

1. $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and*
2. $\mathsf{ipf}$ *is a mapping which associates with each non-leaf object* $o \in V$, *an interval probability function* $\mathsf{ipf}$ *w.r.t.* $\mathsf{potchildren}(o)$, *where* $\mathsf{c} \in \mathsf{potchildren}(o)$ *and* $\mathsf{ipf}(o, \mathsf{c}) = [lb, ub]$.

Intuitively, a probabilistic instance consists of a weak instance, together with probability intervals associated with each potential child of each object in the weak instance. Similarly, given a probabilistic instance, we can obtain its weak instance graph from its corresponding weak instance.

[1] Suppose $\mathbf{S} = \{S_1, \ldots, S_n\}$ where each $S_i$ is a set. A ***hitting set*** for $\mathbf{S}$ is a set $H$ such that (i) for all $1 \leq i \leq n$, $H_i \cap S_i \neq \emptyset$ and (ii) there is no $H' \subset H$ satisfying condition (i).

| $o$ | $l$ | $\mathsf{lch}(o,l)$[a] |
|---|---|---|
| I1 | convoy | { convoy1, convoy2 } |
| convoy1 | tank | { tank1, tank2 } |
| convoy2 | truck | { truck1 } |

| $o$ | $\tau(o)$ | $\mathsf{val}(o)$ |
|---|---|---|
| tank1 | tank-type | T-80 |
| tank2 | tank-type | T-72 |
| truck1 | truck-type | rover |

| $o$ | $l$ | $\mathsf{card}(o,l)$ |
|---|---|---|
| I1 | convoy | [1,2] |
| convoy1 | tank | [ 1,1 ] |
| convoy2 | truck | [ 1,1 ] |

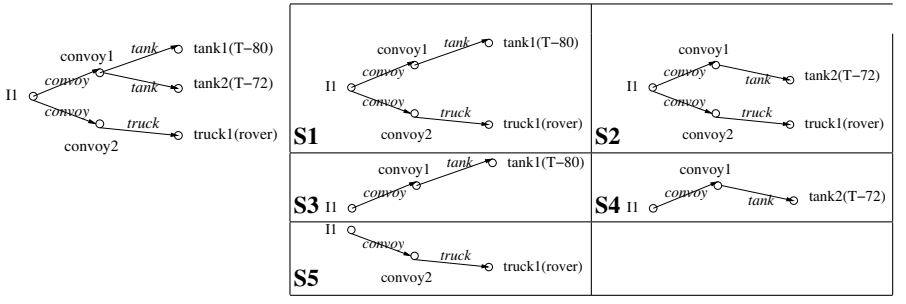
| $c \in \mathsf{potchildren}(I1)$ | $\mathsf{ipf}(I1, c)$ |
|---|---|
| { convoy1} | [ 0.2, 0.4 ] |
| { convoy2} | [ 0.1, 0.4 ] |
| { convoy1, convoy2} | [ 0.4, 0.7 ] |

| $c \in \mathsf{potchildren}(convoy1)$ | $\mathsf{ipf}(convoy1, c)$ |
|---|---|
| { tank1} | [ 0.2, 0.7 ] |
| { tank2} | [ 0.3, 0.8 ] |

| $c \in \mathsf{potchildren}(convoy2)$ | $\mathsf{ipf}(convoy2, c)$ |
|---|---|
| { truck1} | [ 1, 1 ] |

[a] Here we only show objects with non-empty set of children.

**Fig. 2.** A probabilistic instance for the surveillance domain.




(a) (b)

**Fig. 3.** (a) The graph structure of the probabilistic instance in Figure 2. (b) The set of semistructured instances compatible with the probabilistic instance in Figure 2.

*Example 2.* Figure 2 shows a very simple probabilistic instance. The set $\mathcal{O}$ of objects is $\{I1, convoy1, convoy2, tank1, tank2, truck1\}$. The first table shows the legal children of each of the objects, along with their labels. The cardinality constraints are shown in the third table; for example object $I1$ can have 1 to 2 convoy-children. The tables on the right of Figure 2 shows the ipf of each potential child of *I1*, *convoy1* and *convoy2*. Intuitively, $\mathsf{ipf}(I1, \{convoy1\}) = [0.2, 0.4]$ says that the probability of having only *convoy1* is between 0.2 and 0.4.

## 5 *PIXml* : Declarative Semantics

### 5.1 Compatible Semistructured Instances

Recall that any probabilistic instance is defined relative to a weak instance. In this section, we define what it means for a semistructured instance to be compatible with a weak instance. Intuitively, this means that the graph structure of the semistructured instance

is consistent with the graph structure and cardinality constraints of the weak instance. If a given object $o$ occurs in the weak instance $\mathcal{W}$ and $o$ occurs also in a compatible semistructured instance $\mathcal{S}$, then the children of $o$ in $\mathcal{S}$ must be a set of potential children of $o$ in $\mathcal{W}$. We formalize this concept below.

**Definition 7.** *Let $\mathcal{S} = (V_\mathcal{S}, E, \ell, \tau_\mathcal{S}, \mathsf{val}_\mathcal{S})$ be a semistructured instance over a set of objects $\mathcal{O}$, a set of labels $\mathcal{L}$ and a set of types $\mathcal{T}$ and let $\mathcal{W} = (V_\mathcal{W}, \mathsf{lch}_\mathcal{W}, \tau_\mathcal{W}, \mathsf{val}_\mathcal{W}, \mathsf{card})$ be a weak instance. $\mathcal{S}$ is **compatible with** $\mathcal{W}$ if for each $o$ in $V_\mathcal{S}$:*

- *The root of $\mathcal{W}$ is in $\mathcal{S}$.*
- *$o$ is also in $V_\mathcal{W}$.*
- *If $o$ is a leaf in $\mathcal{S}$, then if $o$ is also a leaf in $\mathcal{W}$, then $\tau_\mathcal{S}(o) = \tau_\mathcal{W}(o)$ and $\mathsf{val}_\mathcal{S}(o) = \mathsf{val}_\mathcal{W}(o)$.*
- *If $o$ is not a leaf in $\mathcal{S}$ then*
  - *For each edge $(o, o')$ with label $l$ in $\mathcal{S}$, $o' \in \mathsf{lch}_\mathcal{W}(o, l)$,*
  - *For each label $l \in \mathcal{L}$, $card(o, l).min \leq k \leq card(o, l).max$ where $k = |\{o' | (o, o') \in E \wedge \ell(E) = l\}|$.*

We use $\mathcal{D}(\mathcal{W})$ to denote the set of all semistructured instances that are compatible with a weak instance $\mathcal{W}$. Similarly, for a probabilistic instance $\mathcal{I} = (V, \mathsf{lch}_\mathcal{I}, \tau_\mathcal{I}, \mathsf{val}_\mathcal{I}, \mathsf{card}, \mathsf{ipf})$ we use $\mathcal{D}(\mathcal{I})$ to denote the set of all semistructured instances that are compatible with $\mathcal{I}$'s associated weak instance $\mathcal{W} = (V, \mathsf{lch}_\mathcal{I}, \tau_\mathcal{I}, \mathsf{val}_\mathcal{I}, \mathsf{card})$. Figure 3 shows the set of of all semistructured instances compatible with the weak instance corresponding to the probabilistic instance defined in Example 2.

### 5.2 Global and Local Interpretations

**Definition 8.** *Suppose we have a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$. A **global interpretation** $\mathcal{P}$ is a mapping from $\mathcal{D}(\mathcal{W})$ to $[0, 1]$ such that $\Sigma_{S \in \mathcal{D}(\mathcal{W})} \mathcal{P}(S) = 1$.*

Intuitively, a global interpretation is a distribution over the semistructured instances compatible with a weak instance. On the other hand, local interpretations assign semantics on an object by object basis. To define local interpretations, we first need the concept of an object probability function.

**Definition 9.** *Suppose $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance. Let $o \in V$ be a non-leaf object. An **object probability function** (OPF for short) for $o$ w.r.t. $\mathcal{W}$ is a mapping $\omega : \mathsf{potchildren}(o) \rightarrow [0, 1]$ such that $\Sigma_{\mathsf{c} \in \mathsf{potchildren}(o)} \omega(\mathsf{c}) = 1$.*

**Definition 10.** *Suppose $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance. A **local interpretation** is a mapping $\wp$ from the set of non-leaf objects $o \in V$ to object probability functions, i.e. $\wp(o)$ returns an OPF for $o$ w.r.t. $\mathcal{W}$.*

Intuitively, a local interpretation specifies, for each non-leaf object in the weak instance, an object probability function.

### 5.3 Connections between Local and Global Semantics

In this section, we show that there is a transformation to construct a global interpretation from a local one, and vice versa, and that these transformations exhibit various nice properties.

**Definition 11 ( $\tilde{W}$ operator).** *Let $\wp$ be a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then $\tilde{W}(\wp)$ is a function which takes as input, any $S \in \mathcal{D}(\mathcal{W})$ and is defined as follows: $\tilde{W}(\wp)(S) = \prod_{o \in S} \wp(o)(\mathsf{children}_S(o))$ where $\mathsf{children}_S(o)$ are the actual children of $o$ in $S$.*

The following theorem says that $\tilde{W}(\wp)$ is a valid global interpretation for $\mathcal{W}$ with an acyclic weak instance graph.

**Theorem 2.** *Suppose $\wp$ is a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ with an acyclic weak instance graph. Then $\tilde{W}(\wp)$ is a global interpretation for $\mathcal{W}$.*

**Proof:** (sketch) Define a set of random variables, two for each object $o_i \in V$. One random variable $e_i$ is true *iff* $o_i$ exists in $S$ and false otherwise. $o_i$ will exist if it is a member of the set of children of any node. The second random variable $c_i$ denotes the children of $e_i$. The possible values for $c_i$ are $\mathsf{potchildren}(o_i)$ along with a special null value *null*. If $e_i$ is true, the distribution over $c_i$ is simply $\wp(o)(c)$; if $e_i$ is false $c_i$ is *null* with probability 1.0.

Because our weak instance is acyclic, we can define an ordering for the random variables such that for each $c_i$, $e_i$ occurs before it, and for each $e_i$, any $c_i'$ which is a child set containing $o_i$ occurs before it in the ordering. The distribution can be written as a product of the conditional distributions we have defined above. Furthermore, the distribution that we have just defined is equivalent to $\tilde{W}(\wp)(S)$, hence $\tilde{W}(\wp)(S)$ is a well defined probability distribution.[2] ∎

*Example 3.* Consider the probabilistic instance in Example 2. Suppose we are given a local interpretation $\wp$ such that $\wp(I1) = \omega_{I1}$, $\wp(convoy1) = \omega_{convoy1}$, $\wp(convoy2) = \omega_{convoy2}$, where $\omega_{I1}(\{convoy1\}) = 0.3$. $\omega_{I1}(\{convoy2\}) = 0.2$, $\omega_{I1}(\{convoy1, convoy2\}) = 0.5$, $\omega_{convoy1}(\{tank1\}) = 0.4$, $\omega_{convoy1}(\{tank2\}) = 0.6$, $\omega_{convoy2}(\{truck1\}) = 1$. Then the probability of a compatible instance $S1$ shown in Figure 3 will be $\tilde{W}(\wp)(S1) = \wp(I1)(\{convoy1, convoy2\}) \times \wp(convoy1)(\{tank1\}) \times \wp(convoy2)(\{truck1\}) = 0.5 \times 0.4 \times 1 = 0.2$.

An important question is whether we can go the other way: from a global interpretation, can we find a local interpretation for a weak instance $\mathcal{W}$? It turns out that we can **if** the global interpretation can be factored in a manner consistent with the structure constraints imposed by $\mathcal{W}$. One way to ensure that this is possible is to impose a set of independence constraints relating every non-leaf object and its non-descendants in the weak instance graph $\mathcal{G}_{\mathcal{W}}$. The independence constraints are defined below.

**Definition 12.** *Suppose $\mathcal{P}$ is a global interpretation and $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance. $\mathcal{P}$* **satisfies** *$\mathcal{W}$ iff for every non-leaf object $o \in V$ and each $c \in$*

[2] Essentially what we have done is created a Bayesian network [18] that describes the distribution.

$\mathsf{potchildren}(o)$*, it is the case that*[3]*:* $\mathcal{P}(c|o, \mathsf{ndes}(o)) = \mathcal{P}(c|o)$*. Here,* $\mathsf{ndes}(o)$ *are the non-descendants of* $o$ *in* $\mathcal{G}_{\mathcal{W}}$*.*

In other words, given that $o$ occurs in the instance, the probability of any potential children $c$ of $o$ is independent of any *possible* set of nondescendants. From now on, given a weak instance $\mathcal{W}$, we will only consider $\mathcal{P}$ that satisfies $\mathcal{W}$. The definition below tells us how to associate a local interpretation with any global interpretation.

**Definition 13 ( $\tilde{D}$ operator).** *Suppose* $\mathsf{c} \in \mathsf{potchildren}(o)$ *for some non-leaf object* $o$ *and suppose* $\mathcal{P}$ *is a global interpretation.* $\omega_{\mathcal{P},o}$*, is defined as follows.*

$$\omega_{\mathcal{P},o}(\mathsf{c}) = \frac{\Sigma_{S \in \mathcal{D}(\mathcal{W}) \wedge o \in S \wedge \mathsf{children}_S(o) = \mathsf{c}} \mathcal{P}(S)}{\Sigma_{S \in \mathcal{D}(\mathcal{W}) \wedge o \in S} \mathcal{P}(S)}.$$

*Then,* $\tilde{D}(\mathcal{P})$ *returns a function defined as follows: for any non-leaf object* $o$*,* $\tilde{D}(\mathcal{P})(o) = \omega_{\mathcal{P},o}$*.*

Intuitively, we construct $\omega_{\mathcal{P},o}(\mathsf{c})$ as follows. Find all semistructured instances $S$ that are compatible with $\mathcal{W}$ and given that $o$ occurs, find the proportion for which $o$'s set of children is $\mathsf{c}$. The sum of the (normalized) probabilities assigned to the remaining semistructured instances by $\mathcal{P}$ is assigned to $\mathsf{c}$ by the OPF $\omega_{\mathcal{P},o}(\mathsf{c})$. By doing this for each non-leaf object $o$ and each of its potential child sets, we get a local interpretation. The following theorem establishes this claim formally.

**Theorem 3.** *Suppose* $\mathcal{P}$ *is a global interpretation for a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$*. Then* $\tilde{D}(\mathcal{P})$ *is a local interpretation for* $\mathcal{W}$*.*

*Example 4.* Consider the probabilistic instance in Example 3 and the set of compatible instances in Figure 3. Suppose we are given a global interpretation $\mathcal{P}$ such that $\mathcal{P}(S1) = 0.2$, $\mathcal{P}(S2) = 0.3$, $\mathcal{P}(S3) = 0.12$, $\mathcal{P}(S4) = 0.18$, $\mathcal{P}(S5) = 0.2$. Then a local probability can be obtained by calculating the probability of each potential child of every non-leaf object. For example, when we calculate the probability of $\{tank1\}$ as the actual child of $convoy1$, we notice that $S1, S2, S3, S4$ contain $convoy1$, but only the child of $convoy1$ in $S1, S3$ is $\{tank1\}$. Hence, $\tilde{D}(\mathcal{P})(convoy1)(\{tank1\}) = \frac{\mathcal{P}(S1)+\mathcal{P}(S3)}{\mathcal{P}(S1)+\mathcal{P}(S2)+\mathcal{P}(S3)+\mathcal{P}(S4)} = \frac{0.2+0.12}{0.2+0.3+0.12+0.18} = \frac{0.32}{0.8} = 0.4$

The following theorems tell us that applying the two operators $\tilde{D}$ and $\tilde{W}$ one after the other (on appropriate arguments) yields no change.

**Theorem 4.** *Suppose* $\wp$ *is a local interpretation for a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$*. Then,* $\tilde{D}(\tilde{W}(\wp)) = \wp$*.*

**Theorem 5.** *Suppose* $\mathcal{P}$ *is a global interpretation for a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *and* $\mathcal{P}$ *satisfies* $\mathcal{W}$*. Then,* $\tilde{W}(\tilde{D}(\mathcal{P})) = \mathcal{P}$*.*

[3] Here, $\mathcal{P}(c|o)$ is the probability of $c$ being children of $o$ given that $o$ exists. The notation of $\mathcal{P}(c|o, A)$ means the probability of $c$ being children of $o$ given that $o$ and $A$ exists, where $A$ is a set of objects.

### 5.4 Satisfaction

We are now ready to address the important question of when a local (resp. global) interpretation satisfies a probabilistic instance.

A probabilistic instance imposes constraints on the probability specifications for objects. We associate a set of object constraints with each non-leaf object as follows.

**Definition 14 (object constraints).** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, $o \in V$ is a non-leaf object. We associate with $o$, a set of constraints called **object constraints**, denoted $\mathsf{OC}(o)$, as follows. For each $\mathsf{c} \in \mathsf{potchildren}(o)$, $\mathsf{OC}(o)$ contains the constraint $\mathsf{ipf}(o, \mathsf{c}).lb \leq p(\mathsf{c}) \leq \mathsf{ipf}(o, \mathsf{c}).ub$ where $p(\mathsf{c})$ is a real-valued variable denoting the probability that $\mathsf{c}$ is the actual set of children of $o$. $\mathsf{OC}(o)$ also includes the following constraint $\Sigma_{\mathsf{c} \in \mathsf{potchildren}(o)} p(\mathsf{c}) = 1$.*

*Example 5.* Consider the probabilistic instance defined in Example 2. $\mathsf{OC}(I1)$ is defined as follows: $0.2 \leq p(\{convoy1\}) \leq 0.4$, $0.1 \leq p(\{convoy2\}) \leq 0.4$, $0.4 \leq p(\{convoy1, convoy2\}) \leq 0.7$, and $p(\{convoy1\}) + p(\{convoy2\}) + p(\{convoy1, convoy2\}) = 1$.

Intuitively, an OPF satisfies a non-leaf object iff the assignment made to the potential children by the OPF is a solution to the constraints associated with that object. Obviously, a probability distribution w.r.t. $\mathsf{potchildren}(o)$ over $\mathsf{ipf}$ is a solution to $\mathsf{OC}(o)$.

**Definition 15 (object satisfaction).** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, $o \in V$ is a non-leaf object, $\omega$ is an OPF for $o$, and $\wp$ is a local interpretation. $\omega$ **satisfies** $o$ iff $\omega$ is a probability distribution w.r.t. $\mathsf{potchildren}(o)$ over $\mathsf{ipf}$. $\wp$ satisfies $o$ iff $\wp(o)$ satisfies $o$.*

*Example 6.* Consider the probabilistic instance defined in Example 2, the probability interpretation defined in Example 3 and the $\mathsf{OC}(I1)$ defined in Example 5. Since the assignment made to the potential children of $I1$ by the OPF $\wp(I1) = \omega_{I1}$ is a solution to the constraints $\mathsf{OC}(I1)$ associated with $I1$, $\omega_{I1}$ is a probability distribution w.r.t. $\mathsf{potchildren}(I1)$ over $\mathsf{ipf}$. Thus, $\omega$ satisfies $I1$ and the local interpretation $\wp$ satisfies $convoy$. Similarly, $convoy1$ and $convoy2$ are satisfied.

We are now ready to extend the above definition to the case of satisfaction of a probabilistic instance by a local interpretation.

**Definition 16 (local satisfaction of a prob. inst.).** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, and $\wp$ is a local interpretation. $\wp$ **satisfies** $\mathcal{I}$ iff for every non-leaf object $o \in V$, $\wp(o)$ satisfies $o$.*

*Example 7.* Consider the probabilistic instance defined in Example 2, the local interpretation $\wp$ defined in Example 3. In view of the fact that $\wp$ satisfies all three non-leaf objects, $I1$, $convoy1$ and $convoy2$, it follows that $\wp$ satisfies the example probabilistic instance.

Similarly, a global interpretation $\mathcal{P}$ satisfies a probabilistic instance if the OPF computed by using $\mathcal{P}$ can satisfy the object constraints of each non-leaf object.

**Definition 17 (global satisfaction of a prob. inst.).** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, and $\mathcal{P}$ is a global interpretation. $\mathcal{P}$ satisfies $\mathcal{I}$ iff for every non-leaf object $o \in V$, $\tilde{D}(\mathcal{P})(o)$ satisfies $o$, i.e., $\tilde{D}(\mathcal{P})$ satisfies $\mathcal{I}$.*

**Corollary 1 (equivalence of local and global sat.).** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, and $\wp$ is a local interpretation. Then $\wp$ satisfies $\mathcal{I}$ iff $\tilde{W}(\wp)$ satisfies $\mathcal{I}$.*

We say a probabilistic instance is ***globally (resp. locally) consistent*** iff there is at least one global (resp. local) interpretation that satisfies it. Using Lemma 1, we can prove the following theorem saying that according to our definitions, all probabilistic instances are guaranteed to be globally (and locally) consistent.

**Theorem 6.** *Every probabilistic instance is both globally and locally consistent.*

## 6 *PIXml* Queries

In this section, we define the formal syntax of a ***PIXml*** query. The important concept of a *path expression* defined below plays the role of an attribute name in the relational algebra.

**Definition 18 (path expression).** *A **path expression** $p$ is either an object (oid) $o_{pr}$ or a sequence $o_{pr}.l_1 \ldots l_n$, where $o_{pr}$ is an object (oid), $l_1, l_2, \ldots, l_n$ are labels of edges. If an object $o$ can be located by $p$, then $o \in p$.*

Given an attribute name $A$ in the relational algebra, we can write queries such as $A = v, A \geq v$, etc. We can do the same with path expressions.

**Definition 19 (atomic query).** *An **atomic query** has one of the following forms:*

1. *$p\ \theta\ o$, where $p$ is a path expression, $\theta$ is a binary predicate from $\{=, \neq\}$ and $o$ is an oid.*
2. *$\mathsf{val}(p)\ \theta\ v$, where $p$ is a path expression, $\theta$ is a binary predicate from $\{=, \neq, \leq, \geq, <, >\}$ and $v$ is a value.*
3. *$\mathsf{card}(p, x)\ \theta\ I$, where $p$ is a path expression, $x$ is either $l \in \mathcal{L}$ or $\star$ (a wildcard matches any label), $\theta$ is a binary predicate from $\{=, \neq, \leq, \geq, <, >, \subset, \subseteq, \not\subseteq, \supset, \supseteq, \not\supseteq\}$ and $I$ is an interval. $I_1\ \theta\ I_2$ has the intended interpretation. For example, $I_1 > I_2$ means $I_1.lb > I_2.lb \wedge I_1.ub > I_2.ub$.*
4. *$\mathsf{ipf}(p, x)\ \theta\ I$, where $p$ is a path expression, $x$ is either $\mathsf{c} \in \mathsf{potchildren}(p)$ or the wildcard $\star$ (which means that it matches any potential child), $\theta$ is a binary predicate from $\{=, \neq, \leq, \geq, <, >, \subset, \subseteq, \not\subseteq, \supset, \supseteq, \not\supseteq\}$ and $I$ is an interval $\subseteq [0, 1]$.*
5. *$operand_1\ \theta\ operand_2$, where both of $operand_1$ and $operand_2$ should be of the same form among $p$, $\mathsf{val}(p)$, $\mathsf{card}(p, x)$ and $\mathsf{ipf}(p, x)$ defined above; $\theta$ is a corresponding binary predicate defined above.*

*We assume that an order is defined on the elements in the domain of a type, but some types such as strings only allow operations in $\{=, \neq\}$. An atomic selection expression of the form $\mathsf{val}(p)\ \theta\ v$ or $\mathsf{val}(p_1)\ \theta\ \mathsf{val}(p_2)$ is satisfied only if both sides of the binary predicate are type-compatible and compatible with $\theta$ (i.e., $\theta$ is defined on their types).*

A query is just a boolean combination of atomic queries.

**Definition 20.** *Every atomic query is a* query. *If $q_1, q_2$ are queries, then so are $(q_1 \wedge q_2)$ and $(q_1 \vee q_2)$.*

In order to define the ***answer*** to a query w.r.t. a probabilistic instance, we proceed in three steps. We first define what it means for an object to satisfy a query. We then define what it means for a semistructured instance to satisfy a query. Finally, we define what the answer to a query is w.r.t. a probabilistic instance.

**Definition 21 (satisfaction of a query by an object).** *An object $o_1$ **satisfies** an atomic query $Q$ in the form of $p\ \theta\ o$ (denoted $o_1 \models Q$) if and only if $o_1 \in p \wedge o_1\ \theta\ o$ holds. Similar definitions of satisfaction hold for parts (2)–(4) of the definition of an atomic query.*

*Objects $o_1$ and $o_2$ satisfy an atomic query $p_1\ \theta\ p_2$ if and only if $(o_1 \in p_1 \wedge o_2 \in p_2 \wedge o_1\ \theta\ o_2)$ holds. Similarly for other forms of the two operands.*

Due to space constraints, and because of the need for notational simplicity, we have restricted the above syntax so that only one probabilistic instance is queried at a time. It is straightforward to extend the above syntax to handle multiple probabilistic instances (for example, each of the above queries $Q$ can be prefixed with some notation like $\mathcal{I}_1 : Q$ or $\mathcal{I}_2 : Q$ and the result closed under the boolean operators) to achieve the desired effect. All results in the paper go through with this minor modification and hence we use a simplified syntax here.

In order to define the *answer* to a query, we must account for the fact that a given probabilistic instance is compatible with many semistructured instances. The *probability* that an object is an answer to the query[4] is determined by the probabilities of all the instances that it occurs in.

**Definition 22 (satisfaction of a query by a prob. inst.).** *Suppose $\mathcal{I}$ is a probabilistic instance, $Q$ is a query, and $\mathcal{S} \in \mathcal{D}(\mathcal{I})$. We say that object $o$ **satisfies query** $Q$ **with probability** $r$ **or more**, denoted $o \models^r Q$ iff*

$$r \leq \mathbf{INF}\{\Sigma_{\mathcal{S} \in \mathcal{D}(\mathcal{I}) \wedge o \in \mathcal{S} \wedge o \models Q} \mathcal{P}(\mathcal{S}) \mid \mathcal{P} \models \mathcal{I}\}.$$

Intuitively, any global interpretation $\mathcal{P}$ assigns a probability to each semistructured instance $\mathcal{S}$ compatible with a probabilistic instance $\mathcal{I}$. By summing up the probabilities of the semistructured instances in which object $o$ occurs, we obtain an ***occurrence probability*** for $o$ in $\mathcal{I}$ w.r.t. global interpretation $\mathcal{P}$. However, different global interpretations may assign different probabilities to compatible semistructured instances. Hence, if we examine all global interpretations that satisfy $\mathcal{I}$ and take the **INF** (infimum) of the occurrence probability of $o$ w.r.t. each such satisfying global interpretation, then we are

[4] Here we only consider queries in the form (1) – (4) in Definition 19.

guaranteed that for all global interpretations, the probability of $o$'s occurrence is at least the **INF** obtained in this way. This provides the rationale for the above definition. We may define the answer to a query in many ways. Our standard norm will be to assume that the user sets a real number $0 < r \leq 1$ as a threshold and that only objects satisfying a query with probability $r$ or more will be returned.

**Definition 23 ($r$-answer).** *Suppose $0 < r \leq 1$. The $r$-**answer** to query $Q$ is the set of all objects $o$ such that $o \models^r Q$.*

*Example 8.* Consider the probabilistic instance in Example 2 and a query $Q$ which is $\mathsf{val}(I1.convoy.tank) = T80$. Suppose we want to find a $0.4$-answer to $Q$. Obviously, the only possible object to satisfy $Q$ is $tank1$. However, there exists a global interpretation (for example, the one in Example 4) such that the sum of the probability of compatible instances containing $tank1$ satisfying $Q$ is less than $0.4$. Thus, the $0.4$-answer to $Q$ is empty.

## 7 *PIXml* : Operational Semantics

In this section, we study the problem: ***Given a probabilistic instance $\mathcal{I}$, a real number $0 < r \leq 1$, and a query $Q$, how do we find all objects $o$ in the probabilistic instance that satisfy query $Q$ with probability $r$ or more?***

Clearly, we do not wish to solve this problem by explicitly finding all global interpretations that satisfy $\mathcal{I}$ and explicitly computing the sum on the right side of the inequality in Definition 22. This approach is problematic for many reasons, the first of which is that there may be infinitely many global interpretations that satisfy $\mathcal{I}$.

Recall that the weak instance graph $\mathcal{G}_{\mathcal{W}}$ describes all the potential children of an object. A probabilistic instance is said to be ***tree-structured*** iff its corresponding weak instance graph is a tree (i.e. the vertexes and edges of $\mathcal{G}_{\mathcal{W}}(\mathcal{I})$ must yield a tree rather than a directed acyclic graph). Throughout the rest of this section, we assume that we are only dealing with tree structured probabilistic instances.

Our algorithm to solve this problem involves two core steps.

- **Step 1:** The first step is to identify all objects $o$ in the weak instance graph $\mathcal{G}_{\mathcal{W}}$ of $\mathcal{I}$ that satisfy the query $Q$. This can be done using any available implementation of semistructured databases[16]. Hence we do not present the details here.
- **Step 2:** The second step is to check (using the original probabilistic instance $\mathcal{I}$ rather than its weak instance graph) which of the objects returned in the preceding step have an occurrence probability that exceeds the threshold w.r.t. all global interpretations that satisfy the original probabilistic interpretation.

In this section, we focus on Step 2 as Step 1 can be readily solved using existing techniques for pure, non-probabilistic semistructured databases. To solve step 2, we must have the ability to find the minimal occurrence probability of the type described earlier for $o$ (given an $o$ that passes the test of step 1 above).

Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ and suppose $o \in V$. We define a quantity $\mathsf{cex}(o)$ as follows. Intuitively, $\mathsf{cex}(o)$ finds the ***conditional probability*** of occurrence of $o$, given that $o$'s parent in the tree is known to occur.

1. If $o$ is the root of $\mathcal{G}_{\mathcal{W}}$, then $\mathsf{cex}(o) = 1$.
2. Otherwise, $o \in \mathsf{lch}(o', l)$ for some $o'$ and some $l$. Recall the object constraints $\mathsf{OC}(o')$ defined in Definition 14. Set $\mathsf{cex}(o)$ to be the result of solving the linear programming problem:
   **minimize** $\Sigma_{\mathsf{c} \in \mathsf{potchildren}(o') \wedge o \in \mathsf{c}} p(\mathsf{c})$
   **subject to** $\mathsf{OC}(o')$,
   where $p(\mathsf{c})$ is a real-valued variable denoting the probability that $\mathsf{c}$ is the actual set of children of $o'$. As usual we assume that all the variables $p(\mathsf{c}) \geq 0$.

Intuitively, the quantity $\mathsf{cex}(o)$ specifies the smallest occurrence probability of object $o$ *given that* its parent is already known to occur in a semistructured instance.
**Note:** Though $\mathsf{OC}(o')$ appears to contain exponentially many variables, this is in reality linear in the number of potential children of the object $o'$ and hence linear in the size of $\mathsf{ipf}$ (and hence linear in the size of the probabilistic instance).

**Theorem 7.** *For any probabilistic instance $\mathcal{I}$ and any object $o$, $\mathsf{cex}(o)$ can be computed in time polynomial in the size of $\mathcal{I}$.*

Suppose that the path expression to object $o$ in $\mathcal{I}$ is of the form $r.l_1 \ldots l_n$ where $r$ is the root, and $l_1, \ldots, l_n$ are labels. Let us now use the object id $o_n$ to denote $o$, and object id $o_{i-1}$ to denote the parent of $o_i$ where $1 \leq i \leq n$. Then the ***computed occurrence probability,*** $\mathsf{cop}(o)$ ***of*** $o$ ***in*** $\mathcal{I}$ is given by $\mathsf{cex}(o_0) \cdots \mathsf{cex}(o_n)$.

The following theorem says that the occurrence probability of $o$ (which is defined declaratively in Definition 22) corresponds exactly to the computed occurrence probability of $o$ according to the above procedure.

**Theorem 8.** *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a tree structured probabilistic instance and $o \in V$. Then,* $\mathsf{cop}(o) = \mathbf{INF}\{\Sigma_{\mathcal{S} \in \mathcal{D}(\mathcal{I}) \wedge o \in \mathcal{S}} \mathcal{P}(S) \mid \mathcal{P} \models \mathcal{I}\}$.

**Proof** (sketch) We provide a detailed sketch of the proof of this theorem. Suppose that the path expression to object $o$ in $\mathcal{I}$ is of the form $r.l_1 \ldots l_n$ where $r$ is the root's object id, and $l_1, \ldots, l_n$ are labels. Let us now use the object id $o_n$ to denote $o$, and object id $o_{i-1}$ to denote the parent of $o_i$ where $1 \leq i \leq n$. The full proof (which is only summarized here) proceeds by induction on $n$.

If $n = 0$ then $o$ is the root of $\mathcal{I}$. Hence, $\mathsf{cop}(o) = 1$ by definition. As every compatible semistructured instance contains the root, it follows by the definition of probabilistic interpretation that every probabilistic interpretation assigns 1 to $\Sigma_{\mathcal{S} \in \mathcal{D}(\mathcal{I}) \wedge o \in \mathcal{S}} \mathcal{P}(S)$. This completes the base case.

Suppose $n = m+1$ for some integer $m$. Consider the path $o_1.l_2 \ldots l_m l_{m+1}$. Clearly, this is a path of length $m$ from $o_1$ to $o_{m+1} = o$. Let $\mathcal{I}'$ be the probabilistic instance which is just like the subtree (of $\mathcal{I}$) rooted at $o_1$. By the induction hypothesis, we know that $\mathsf{cop}_{\mathcal{I}'}(o) = \Sigma_{\mathcal{S}' \in \mathcal{D}(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')$ where $\mathcal{P}'$ be defined as follows. If $\mathcal{S}'$ is compatible with $\mathcal{I}'$ then $\mathcal{P}'(\mathcal{S}')$ is the sum of $\mathcal{P}(\mathcal{S})$ for all $\mathcal{S}$ compatible with $\mathcal{I}$ such that $\mathcal{S}'$ is the subtree of $\mathcal{S}$ rooted at $o_1$.

Note that $\mathsf{cop}_{\mathcal{I}}(o) = \mathsf{cex}(o_0) \times \mathsf{cop}_{\mathcal{I}'}(o_1)$ by definition of $\mathsf{cop}$. From the above, we may infer that $\mathsf{cop}(o) = \mathsf{cex}(o_0) \times \Sigma_{\mathcal{S}' \in \mathcal{D}(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')$. We therefore need to show that

$$\mathsf{cex}(o_0) \times \Sigma_{\mathcal{S}' \in \mathcal{D}(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S') \text{ equals } \Sigma_{\mathcal{S} \in \mathcal{D}(\mathcal{I}) \wedge o \in \mathcal{S} \wedge \mathcal{P} \models \mathcal{I}} \mathcal{P}(S).$$

But $\mathsf{cex}(o_0) = 1$ by definition, so we need to show that

$$\Sigma_{\mathcal{S}' \in \mathcal{D}(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models_{\mathcal{I}'} \mathcal{P}'(S')} = \Sigma_{\mathcal{S} \in \mathcal{D}(\mathcal{I}) \wedge o \in \mathcal{S} \wedge \mathcal{P} \models_{\mathcal{I}} \mathcal{P}(S)}.$$

The reason this equality holds is because of the construction of $\mathcal{P}'$. Consider each compatible instance $\mathcal{S} \in \mathcal{D}(\mathcal{I})$ that contains $o$. The interpretation $\mathcal{P}'$ looks at each $\mathcal{S}'$ obtained by restricting $\mathcal{S}$ to the subtree rooted at $o_1$. A single $\mathcal{S}$ generates only one $\mathcal{S}'$ in this way, but the same $\mathcal{S}'$ may be obtained from different semistructured instances $\mathcal{S}$ compatible with $\mathcal{I}$. The sum of all such $\mathcal{P}'(\mathcal{S}')$ clearly equals the sum of all such $\mathcal{P}(\mathcal{S})$ containing $o$ by construction. Furthermore, $\mathcal{P}$ satisfies $\mathcal{I}$ iff its corresponding $\mathcal{P}$" satisfies $\mathcal{I}'$. This completes the sketch of the proof. ∎

It is important to note that the above condition is very similar to the condition on the right side of the inequality of Definition 22 (it is exactly the same if we only consider objects $o$ that satisfy the query). ***An immediate consequence of the above theorem is that the two-step procedure outlined in this section is correct, as step 2 is only applied to objects that satisfy the query condition*** $Q$***.***

## 8 Related Work

Dekhtyar et. al. [5] were the first to deal with probabilities and semistructured data. They pioneered the integration of probabilities and semistructured data by introducing the concept of a structured probability object (SPO). An SPO is a way of representing a collection of random variables[20]. They introduce the concept of a context — intuitively, contexts provide information about when a probability distribution is applicable. They develop an algebra to query databases of such SPOs and a prototype implementation. Here, we do not develop an algebra; the query language described in this paper is a very simple logical one. We also provide a model theory (two in fact) and an operational semantics and show that our operational semantics is correct.

More recently, Nierman et. al.[17] developed a framework called ProTDB to extend the XML data model to include probabilities. XML DTD's are extended to use a `Prob` attribute for each element. Unlike the ***PIXml*** approach, they assume that the probabilities of edges are conditional probabilities which are *independent of the other children of a node*. Their answers are correct under the assumption of conditional independence and under the condition that the underlying graph is tree-structured. Our approach is somewhat different. First, we allow arbitrary correlations between a node's children to be expressed; thus our model subsumes their model. Second, we use an *interval probability* model rather than a point probability model. This is useful because almost all statistical evidence involves margins of error. So when a statistical estimate says that something is true with probability $95\%$ with a $\pm 2\%$ margin of error, then this really corresponds to saying the event's probability lies in the interval $[0.93, 0.97]$. Likewise, using intervals is valuable when one does not know the relationship between different events. For example, if we know the probabilities of events $e_1, e_2$ and want to know the probability of both of them holding, then we can, in general, only infer an interval for the conjunction of $e_1, e_2$ (cf.Boole[2,8]) *unless* we know something more about the dependencies or lack thereof between the events. Third, we provide two formal declarative semantics for probabilistic semistructured databases - no such model theory is proposed by [17]. We additionally

prove that these model theories have a variety of interesting relationships. Finally, we prove that our ***PIXml*** query language is sound and complete w.r.t. the above model theory.

The above two pieces of work are closest to ours. In addition, there has been extensive work on probabilistic databases in general. Kiessling et. al.'s DUCK system [9, 12] provides a logical, axiomatic theory for rule based uncertainty. Lakshmanan and Sadri [14] show how selected probabilistic strategies can be used to extend the previous probabilistic models. Lakshmanan and Shiri [15] have shown how deductive databases may be parameterized through the use of conjunction and disjunction strategies. Barbara et al. [1] develop a point probabilistic data model and propose probabilistic operators. Cavallo and Pittarelli [4]'s important probabilistic relational database model uses probabilistic projection and join operations, but the other relational algebra operations are not specified. Also, a relation in their model is analogous to a single tuple in the framework of [1]. Dey and Sarkar [6] propose an elegant 1NF approach to handling probabilistic databases. They support (i) having uncertainty about some objects but certain information about others, (ii) first normal form which is easy to understand and use, (iii) new operations like conditionalization. The ProbView system by Lakshmanan et. al. [13] extends the classical relational algebra by allowing users to specify in their query what probabilistic strategy (or strategies) should be used to parameterize the query. ProbView removes the independence assumption of previous works. More recently, Dyreson and Snodgrass pioneered the use of probabilities over time in databases[7].

## 9 Conclusion

Nierman et. al.[17] have argued eloquently for the need to represent uncertainty in semistructured data of applications like information retrieval and protein chemistry applications. There are many other applications ranging from image surveillance applications to the management of predictive information.

In this paper, we have developed a formal theory for probabilistic semistructured data. Specifically, we have shown how graph models of semistructured data may be augmented to include probabilistic information. In addition, we have provided two formal declarative semantics for such databases. The first is a global semantics that draws inspiration from classical probabilistic model theory as exemplified in the work of Fagin et. al.[8] and applies it to the probabilistic XML model proposed here. We also propose a local semantics that may be more intuitive in many respects. We show rich correspondences between the two semantics. We propose a query language to query such sources and provide an operational semantics that is proven to be sound and complete. To our knowledge, the declarative semantics and the soundness and completeness results are the first of their kind.

**Acknowledgement.** Edward Hung is supported by the Croucher Foundation Scholarships. Work is funded in part by the Army Research Lab under contract number DAAL0197K0135, the Army Research Office under grant number DAAD190010484, by DARPA/RL contract number F306029910552, the ARL CTA on Advanced Decision Architectures, and NSF grant 0205489.

## References


1. D. Barbara, H. Garcia-Molina and D. Porter. (1992) The Management of Probabilistic Data, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, pp 487–502.
2. G. Boole. (1854) *The Laws of Thought*, Macmillan, London.
3. B. Bouwerman and R.T. O'Connell. (2000) Forecasting and Time Series: An Applied Approach, Brooks/Cole Publishing.
4. R. Cavallo and M. Pittarelli. (1987) The Theory of Probabilistic Databases, in *Proc. VLDB'87.*
5. A. Dekhtyar, J. Goldsmith and S.R. Hawkes. (2001) *Semi-structured Probabilistic Databases*, Proceedings of 2001 Conference on Statisitcal and Scientific Database Management (SSDBM), George Mason University, Fairfax, VA, USA, pp. 36–45, July 2001.
6. D. Dey and S. Sarkar. (1996) A Probabilistic Relational Model and Algebra, *ACM Transactions on Database Systems*, Vol. 21, 3, pp 339–369.
7. C. Dyreson and R. Snodgrass. (1998) Supporting Valid-Time Indeterminacy, *ACM Transactions on Database Systems*, Vol. 23, Nr. 1, pp 1–57.
8. Fagin, R., J. Y. Halpern, and N. Megiddo (1990). A logic for reasoning about probabilities. *Information and Computation 87*(1/2), 78–128.
9. U. Guntzer, W. Kiessling and H. Thone. (1991) *New Directions for Uncertainty Reasoning in Deductive Databases*, Proc. 1991 ACM SIGMOD, pp 178–187.
10. E. Hung, L. Getoor, V.S. Subrahmanian. (2003) *PXML: A Probabilistic Semistructured Data Model and Algebra*, Proc. of the 19th International Conference on Data Engineering, Bangalore, India, March 5–8, 2003.
11. G. Kamberova and R. Bajcsy. (1998) Stereo Depth Estimation: the Confidence Interval Approach, Proc. Intl. Conf. Computer Vision ICCV98, Bombay, India, Jan. 1998
12. W. Kiessling, H. Thone and U. Guntzer. (1992) *Database Support for Problematic Knowledge*, Proc. EDBT-92, pp. 421–436, Springer LNCS Vol. 580.
13. V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian. (1997) ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems*, Vol. 22, Nr. 3, pp. 419–469.
14. V.S. Lakshmanan and F. Sadri. (1994) Probabilistic Deductive Databases, in *Proc. Int. Logic Programming Symp., (ILPS'94)*, MIT Press.
15. V.S. Lakshmanan and N. Shiri. (1996) Parametric Approach with Deductive Databases with Uncertainty, in *Proc. Workshop on Logic In Databases 1996*, pp. 61–81.
16. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. (1997) *Lore: A Database Management System for Semistructured Data*, ACM SIGMOD Record, Sep. 1997.
17. A. Nierman and H.V. Jagadish. (2002) *ProTBD: Probabilistic Data in XML*, in Proc. of the 28th International Conference on Very Large Data Bases, Hong Kong, August, 2002.
18. J. Pearl. (1988) *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann.
19. D.Radev, W. Fan and H. Qi. (2002) *Probabilistic question answering from the Web*, Proc. 11th International World Wide Web Conference, Hawaii, May 2002.
20. S. Ross. (1998) *A First Course in Probability*, Prentice Hall, 1998.
21. *Extensible Markup Language (XML) (2002)*, W3C. Available at `http://www.w3.org/XML/`.

# Condensed Representation of Database Repairs for Consistent Query Answering

Jef Wijsen

Université de Mons-Hainaut (UMH), Institut d'Informatique
Avenue du Champ de Mars, 6, B-7000 Mons, Belgium
jef.wijsen@umh.ac.be

**Abstract.** Repairing a database means bringing the database in accordance with a given set of integrity constraints by applying modifications that are as small as possible. In the seminal work of Arenas et al. on query answering in the presence of inconsistency, the possible modifications considered are deletions and insertions of tuples. Unlike earlier work, we also allow tuple updates as a repair primitive. Update-based repairing is advantageous, because it allows rectifying an error within a tuple without deleting the tuple, thereby preserving other consistent values in the tuple. At the center of the paper is the problem of query answering in the presence of inconsistency relative to this refined repair notion. Given a query, a trustable answer is obtained by intersecting the query answers on all repaired versions of the database. The problem arising is that, in general, a database can be repaired in infinitely many ways. A positive result is that for conjunctive queries and full dependencies, there exists a condensed representation of all repairs that permits computing trustable query answers.

## 1 General Problem Description

In most database textbooks, databases that violate integrity constraints are considered as "illegal" and hence disregarded. Nevertheless, databases containing inconsistencies are common in real life, and it is important to understand how to treat this phenomenon. At the time inconsistencies are detected, "cleaning" the database is often problematic, as there is generally not one single deterministic way of rectifying inconsistencies. The least we can do is prohibit inconsistencies from being propagated into query answers visible by end-users.

For example, to avoid another dioxin-in-food crisis, the government is continuously watching dioxin levels in food samples. The first tuple of the dioxin database in Fig. 1 says that the sample 110 was taken on 17 Jan 2002 and analyzed the day after by the ICI lab, which found a normal dioxin level. The obvious integrity constraint *"The sample date must be prior to the analysis date"* is violated by sample 220. This inconsistency can be "cleaned" in several ways, for example, by antedating the sample date or by postdating the analysis date of sample 220.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 378–393, 2003.

| *Dioxin* | *SampleNum* | *SampleDate* | *Food* | *AnalysisDate* | *Lab* | *DioxinLevel* |
|---|---|---|---|---|---|---|
| | 110 | 17 Jan 2002 | poultry | 18 Jan 2002 | ICI | normal |
| | 220 | 17 Jan 2002 | poultry | 16 Jan 2002 | ICB | alarming |
| | 330 | 18 Jan 2002 | beef | 18 Jan 2002 | ICB | normal |

**Fig. 1.** Dioxin database.

A characterization of consistent query answer relies on the notion of repair. For simplicity, we focus on unirelational databases. Let $\mathbf{I}$ be the set of all relations of a fixed arity $n$. The following discussion is relative to some class $\mathbf{Q}$ of queries and some class $\mathbf{C}$ of constraints. Let $\Sigma \in \mathcal{P}^{\text{fin}}(\mathbf{C})$ be a satisfiable set of constraints.[1] Intuitively, a *repair* for a relation $I$ and $\Sigma$ is a relation that satisfies $\Sigma$ and that can be obtained from $I$ by applying some "minimal change." Defining "minimal change" is a main theme of this paper; for now simply assume we have some definition that allows determining repairs. In general, the number of repairs for $I$ and $\Sigma$ can be large or even infinite. We say that a tuple is a consistent answer to some query if it is in the answer to the query on each possible repair. Formally, a tuple $t$ is a *consistent answer* to some query $Q$ on input $I$ and $\Sigma$ if for every repair $I'$ for $I$ and $\Sigma$, $t \in Q(I')$.

Previous work on consistent query answering [2,3,4,5,7,10,11,13] assumes that tuple insertions and deletions are the basic primitives for repairing an inconsistent database. We call this approach *tuple-based* repairing, because it treats tuples as atomic. Following this approach, there is a unique repair for the *Dioxin* relation, which is obtained by simply deleting the faulty tuple about sample 220. The query *"Get numbers of alarming samples"* will then yield the empty consistent answer. Nevertheless, it seems more reasonable to conclude that, yes, there was a sample 220 with an alarming dioxin level, but either the sample or the analysis date contains an error. The problem with tuple-based repairing is that an entire tuple may be deleted, even if only a minor part of it is erroneous. There is a need for finer repair primitives that enable correcting faulty values within a tuple, without deleting the tuple. In this paper, we propose an approach capable of repairing at the attribute-value level, which will be called *value-based* repairing.

Whatever definition of repair is adopted, the notion of consistent answer talks about all possible repairs, and hence is impractical, since the number of repairs can be very large or even infinite. So a remaining task is to develop effective methods to compute consistent query answers. Arenas et al. [3] propose a "query transformation" technique for computing consistent answers. Given a satisfiable set $\Sigma \in \mathcal{P}^{\text{fin}}(\mathbf{C})$ and a query $Q$, they apply a query transformation $f_\Sigma : \mathbf{Q} \to \mathbf{Q}'$, where the target query class $\mathbf{Q}'$ may be different from $\mathbf{Q}$, such that for every relation $I$, $f_\Sigma(Q)(I)$ yields exactly the consistent answers to $Q$ on input $I$ and $\Sigma$. The practical intuition is that the database $I$ is left as is and queries are transformed through $f_\Sigma$ when they arrive, in such a way that the transformed query on the original, possibly inconsistent, database yields exactly

[1] $\mathcal{P}^{\text{fin}}(\cdot)$ is the finite powerset operator.

the consistent answers. Because the repair work is deferred until query time, we call this approach *late-repairing*.

In this paper, we study the suitability of "database transformation" as an alternate technique for late-repairing: given a satisfiable set $\Sigma \in \mathcal{P}^{\text{fin}}(\mathbf{C})$ and a relation $I$, apply a database transformation $h_\Sigma : \mathbf{I} \to \mathbf{I}$ such that for every query $Q$, $Q(h_\Sigma(I))$ yields exactly the consistent answers to $Q$ on input $I$ and $\Sigma$. Importantly, $h_\Sigma(I)$ itself need not be a repair for $I$ and $\Sigma$, so the intended database transformation is different from computing repairs, which was argued to be impractical; one may think of $h_\Sigma(I)$ as a "condensed representation" of all possible repairs for $I$ and $\Sigma$ that is sufficient for consistent query answering. The practical intuition is that an inconsistent database $I$ is first transformed through $h_\Sigma$ in such a way that subsequent queries on the transformed database retrieve exactly the consistent answers. Since databases are modified prior to query execution, we call this approach *early-repairing*.

In many database applications, the set $\Sigma$ of integrity constraints will not change. Then, from a practical viewpoint, early-repairing is advantageous in situations where many *ad hoc* queries are asked on a fixed database $I$: the database is transformed once through $h_\Sigma$ and then produces consistent answers to any query. Conversely, late-repairing is advantageous if database updates are frequent and queries are known in advance: the queries are transformed once through $f_\Sigma$ and the transformed queries are used afterwards to obtain consistent answers on a changing database.

The main contributions of this paper can now be summarized as follows. We provide a general framework for repairing and then put into the details so as to arrive at the novel notion of value-based repairing. We show the feasibility of value-based early-repairing for consistent query answering by determining a database transformation $h_\Sigma$ if constraints are full dependencies and queries are conjunctive.

The paper is organized as follows. The construct of value-based repair is settled in Sections 2 through 4. Section 5 defines the notion of consistent (or trustable) query answer. Sections 6 through 9 consider effectively computing consistent answers for conjunctive queries and full dependencies. The computation relies on greatest lower bounds of tableaux under $\theta$-subsumption (Section 7) and on the chase technique (Section 8). Section 10 contains some initial results on the complexity of the proposed technique. Section 11 relates our main results to existing work, and describes some problems for further research.

## 2 General Repairing Framework

Fixing (or repairing) a relation $I$ with respect to a set $\Sigma$ of constraints means modifying $I$ to bring it in accordance with $\Sigma$. We assume that repairing is done relative to some partial order, denoted $\geq$, on relations. Several orders may be appropriate for the purpose of repairing, and some choices will be discussed hereafter. We start from a relation $I$ that may violate a set $\Sigma$ of constraints. We say that a relation $F$ *subsatisfies* $\Sigma$ if there exists a relation $J \geq F$ such that

$J \models \Sigma$. More precisely, $F$ need not be a classical relation; in our approach, $F$ can be a tableau containing variables. *Downfixing* means that we pass from $I$ to $F$ such that $I \geq F$ and $F$ subsatisfies $\Sigma$. *Upfixing* means that we subsequently pass from $F$ to a relation $M \geq F$ such that $M \models \Sigma$. We look at *fixing* as downfixing followed by upfixing. In order to avoid confusion with the repair construct in [3], the final result $M$ of fixing will be called a *mend*, and the intermediate result $F$ is called a *fix*.

It is natural to require that in fixing a relation, we must only apply modifications that are as small as possible. In this way, we ensure that the result of fixing is as close as possible to the initial relation. This "minimal change" or "maximal closeness" principle can be settled by two different criterions:

1. *Maximal content preservation:* Downfixing retains as much as possible from the original relation, and upfixing consists of a minimal completion: $F$ should be such that there exists no $F'$ that also subsatisfies $\Sigma$ and such that $I \geq F' > F$ (i.e., $F'$ is "closer" to $I$ than $F$). Next, for a given $F$, $M$ should be such that there exists no $M'$ such that $M > M' \geq F$ and $M' \models \Sigma$. Note that this criterion solely relies on the order $\geq$. It is similar in spirit to the minimization of tuple deletions in [12].
2. *Minimal distance fixing:* Set containment $\supseteq$ is probably the most obvious choice for the order $\geq$. Under $\supseteq$, downfixing simply means deleting tuples from $I$, and upfixing simply means inserting tuples into $F$. Under the order $\supseteq$, it is natural to minimize (w.r.t. $\subseteq$) the symmetric difference between $I$ and $M$, i.e., $(I \setminus M) \cup (M \setminus I)$. The part $(I \setminus M)$ contains the tuples deleted during downfixing, and $(M \setminus I)$ the tuples added during upfixing. In this case, $F$ corresponds to $I \setminus (I \setminus M) = I \cap M$. This approach is taken in [3]. [13] minimizes the cardinality of the symmetric difference instead.

Under both criterions, the intermediate fix $F$ and the target mend $M$ will generally not be unique. It is easy to verify that under $\supseteq$, every mend found under criterion 1 is a mend under criterion 2, but the inverse is not true.

If we choose $\supseteq$ for the order $\geq$, then we cannot differentiate between tuples that agree on most attributes and tuples that disagree on all attributes: the tuples will simply be treated as unequal in both cases; hence, the repairing will be tuple-based. In order to achieve value-based repairing, which was argued to be more attractive in Section 1, we allow tableaux for the intermediate result $F$ and use $\theta$-subsumption for the order $\geq$. A tableau $T$ is said to $\theta$-subsume a tableau $S$, denoted $T \succeq S$, if there exists a substitution $\theta$ such that $\theta(S) \subseteq T$. The substitution $\theta$ is commonly called a homomorphism from $S$ to $T$ in the database community, and is intimately related to query containment [1] and to logical implication [14]. While $\theta$-subsumption is commonly considered between clauses, we use it between tableaux representing the negation of a clause: the tableau $\{t_1, \ldots, t_m\}$ can be treated as $\exists^*(t_1 \wedge \ldots \wedge t_m)$, i.e., as the negation of the clause $\forall^*(\neg t_1 \vee \ldots \vee \neg t_m)$.

Clearly, $T \supseteq S$ implies $T \succeq S$; hence, $\theta$-subsumption weakens the order $\supseteq$, which was argued to be too rough for repairing purposes. Using $\theta$-subsumption has the effect of allowing updates as repair primitive: downfixing can replace a

constant by a variable, which is subsequently replaced by a different constant during upfixing.

If tuples can be updated in place, then the symmetric difference provides no longer a good criterion for the distance between a database and its repairs, for the simple reason that symmetric difference has the undesirable effect of treating a tuple modification as a combination of a deletion and an insertion. Instead of establishing a new, adjusted distance measure between a database and its repairs, we prefer adhering to the principle of *"Maximal content preservation,"* which solely relies on the underlying order $\geq$.

## 3 Downfixing Revisited

Downfixing based on $\theta$-assumption is attractive from a logic perspective. Intuitively, downfixing replaces untrustworthy values in a relation by existentially quantified variables while ensuring that the result is logically implied by the original relation. Nevertheless, $\theta$-subsumption for downfixing raises a technical as well as a semantical problem, which, fortunately, can be gracefully solved.

A technical problem is the non-existence of greatest fixes under $\succeq$ in certain situations. For example, let $I = \{\langle a, a\rangle\}$ and $\Sigma = \{\forall x(\neg R(x, x))\}$. From Corollary 3.3 in [16], it follows that there exists an infinite sequence $F_1, F_2, \ldots$ of tableaux such that (i) $I \succ \ldots \succ F_2 \succ F_1$, (ii) for each $i \in \mathbb{N}$, $F_i$ subsatisfies $\Sigma$, and (iii) there exists no tableau $F$ such that $F$ subsatisfies $\Sigma$ and for each $i \in \mathbb{N}$, $F \succeq F_i$. It follows that the principle of maximal content preservation is not generally applicable.

A semantical problem is that $\theta$-subsumption turns out to be too weak, as it accepts mends with spurious tuples. For example, let $\Sigma = \{\forall x, y(R(x, y) \rightarrow x = y)\}$. Let $I = \{\langle a, b\rangle\}$, $F = \{\langle a, y\rangle, \langle x, b\rangle\}$, and $M = \{\langle a, a\rangle, \langle b, b\rangle\}$. We have $I \not\models \Sigma$. The valuation $\omega = \{x/a, y/b\}$ yields $\omega(F) = I$, hence $I \succeq F$. Next, the valuation $\theta = \{x/b, y/a\}$ yields $\theta(F) = M$, hence $M \succeq F$. Since $M \models \Sigma$, $F$ subsatisfies $\Sigma$. One can verify that $F$ would be a legal fix, and $M$ a legal mend. Intuitively, the faulty tuple $\langle a, b\rangle$ in $I$ is repaired by both $\langle a, y\rangle$ and $\langle x, b\rangle$ in $F$, which results in two tuples in $M$, either of which is somehow spurious. It would be more natural to assume that either the first or the second coordinate of $\langle a, b\rangle$ is erroneous, but not to combine both assumptions in a single fix. That is, it would be more natural to end up with two separate mends $M_1 = \{\langle b, b\rangle\}$ and $M_2 = \{\langle a, a\rangle\}$.

We have argued in the preceding sections that $\supseteq$ is too strong for repairing, as it does not allow updating tuples. It now turns out that $\succeq$ is too weak for downfixing, as it can result in mends with spurious tuples. Therefore, downfixing will be based on a relation in between $\supseteq$ and $\succeq$. In particular, it will be required that no tuple of $I$ be fixed more than once in $F$.

## 4 Fixes and Mends

We are now ready to formalize the constructs introduced so far.

**Definition 1.** *We assume two disjoint, infinite sets* **dom** *and* **var** *of* constants *and* variables. *A* symbol *is either a constant or a variable. A* tuple *of* arity $n$ *(or $n$-tuple) is a sequence $\langle p_1, \ldots, p_n \rangle$ of symbols ($n \in \mathbb{N}$). A* tableau *of* arity $n$ *is a finite set of $n$-tuples. Unless stated otherwise, we henceforth assume a fixed arity $n$ for tuples and tableaux; the set of all such tableaux is denoted* $\mathbf{T}$. *The cardinality of a tableau $T$ is denoted $|T|$. A tuple is* ground *if it contains no variables; a tableau is* ground *if all its tuples are ground. A* relation *(of arity $n$) is a ground tableau. The set of all relations is denoted* $\mathbf{I}$. *If $T$ is a tableau, then* $\mathbf{grd}(T)$ *denotes the smallest relation that contains every ground tuple of $T$.*

*We write id for the identity function on symbols; and $id_{p=q}$, where $p$ and $q$ are not two distinct constants, for a substitution that identifies $p$ and $q$ and that is the identity otherwise. That is, if $p$ is a variable and $q$ a constant, then $id_{p=q} = \{p/q\}$; similar, mutatis mutandis, if $p$ is a constant and $q$ a variable. If $p$ and $q$ are variables, then $id_{p=q}$ can be either $\{p/q\}$ or $\{q/p\}$; the choice will be immaterial in the technical treatment.*

*Let $T, S$ be two tableaux (of the same arity $n$). $S$ is said to* $\theta$-subsume *$T$, denoted $S \succeq T$, iff there exists a substitution $\theta$ such that $\theta(T) \subseteq S$. We write $S \sim T$ iff $S \succeq T$ and $T \succeq S$. We write $S \succ T$ iff $S \succeq T$ and $S \not\sim T$. The relation $\sim$ is an equivalence relation (Lemma 1); we write $[T]_\sim$ for the equivalence class of $\sim$ that contains $T$. A tableau is* minimal *if it is $\sim$-equivalent to no tableau of smaller cardinality.*

As motivated in Section 3, downfixing will be based on a relation $\sqsupseteq$ whose "strength" lies between $\supseteq$ and $\succeq$.

**Definition 2.** *Let $T, S$ be tableaux of the same fixed arity (i.e., $T, S \in \mathbf{T}$). We write $S \sqsupseteq T$ iff there exists a substitution $\theta$ such that $\theta(T) \subseteq S$ and $|\theta(T)| = |T|$. The latter condition says that $\theta$ must not identify distinct tuples of $T$.*

**Lemma 1.** *Let $T, S$ be tableaux in* $\mathbf{T}$.

1. *If $S \supseteq T$, then $S \sqsupseteq T$. If $S \sqsupseteq T$, then $S \succeq T$.*
2. *If $S \succeq T$, then $\mathbf{grd}(S) \supseteq \mathbf{grd}(T)$.*
3. *$\langle \mathbf{T}, \succeq \rangle$ and $\langle \mathbf{T}, \sqsupseteq \rangle$ are quasi-orders.*[2]
4. *$\sim$ is an equivalence relation on* $\mathbf{T}$.

We now formalize the constructs of fix and mend, adhering to the principle of *"Maximal content preservation"* presented in Section 2.

**Definition 3.** *Let $I$ be a relation (of arity $n$) and $\Sigma$ a set of FO sentences. Since our focus is on unirelational databases, $\Sigma$ is expressed in a first-order (FO) language using a single $n$-ary predicate symbol.*

*A tableau $T$ is said to* subsatisfy *$\Sigma$ iff there exists a relation $J \succeq T$ such that $J \models \Sigma$. A* fix *for $I$ and $\Sigma$ is a tableau $F$ such that $I \sqsupseteq F$, $F$ subsatisfies $\Sigma$, and for every tableau $F'$, if $I \sqsupseteq F' \succ F$, then $F'$ does not subsatisfy $\Sigma$.*

*A* mend *for $I$ and $\Sigma$ is a relation $M$ with $M \models \Sigma$ such that there exists a fix $F$ for $I$ and $\Sigma$ satisfying: (i) $M \succeq F$ and (ii) for every relation $M'$, if $M \succ M' \succeq F$, then $M' \not\models \Sigma$.*

[2] A quasi-order satisfies reflexivity and transitivity.

The requirement $I \sqsupseteq F$ in the above definition implies the existence of a substitution $\theta$ such that $\theta(F) \subseteq I$ and $|\theta(F)| = |F|$. This solves the conceptual problem raised in Section 3: for a given tuple $t \in I$, there can be at most one "repairing" tuple $t' \in F$ such that $\theta(t') = t$. The technical problem is also solved: as $|F| \leq |I|$ is obvious, the number of non-equivalent (under $\sim$) candidate fixes is finite.

**Lemma 2.** *Let $I$ be a relation, and $\Sigma$ a set of FO sentences.*

1. *The set $\{[F]_\sim \mid F \in \mathbf{T}$ and $I \sqsupseteq F\}$ is finite.*
2. *If $\Sigma$ is satisfiable, then there exists a fix for $I$ and $\Sigma$.*

The above result does not mean that determining fixes is feasible in general. Indeed, it is easy to show that the problem of determining whether a given tableau $F$ subsatisfies a given satisfiable set $\Sigma$ of FO sentences, is undecidable.

*Example 1.* In the dioxin example, one fix (call it $F_1$) is the tableau obtained from the relation *Dioxin* by substituting $x$ for the sample date of sample 220. A second fix (call it $F_2$) is the tableau obtained by substituting $y$ for the analysis date of sample 220. All other fixes are equal to $F_1$ or $F_2$ up to a renaming of variables.

Every mend is obtained either from $F_1$ by replacing $x$ by any date not after 16 Jan 2002, or from $F_2$ by replacing $y$ by any date not before 17 Jan 2002. If we assume unbounded time, then there are infinitely many mends, each of which contains an alarming sample.

## 5 Trustable Query Answers

For a given relation and set of constraints, the number of mends can be infinite. The problem of querying all these mends together to obtain consistent answers, is at the center of this paper. In order to avoid confusion with the consistency notion of [3], we use the term "trustable" instead of consistent in the technical treatment.

**Definition 4.** *Let $I$ be a relation (of arity $n$), $\Sigma$ a satisfiable set of FO sentences, and $Q$ a query of some fixed query class. Since our focus is on unirelational databases, the input of each query considered is an $n$-ary relation. Then the ground tuple $t$ is a* trustable answer *to $Q$ on input $I$ and $\Sigma$ iff $t \in Q(M)$ for every mend $M$ for $I$ and $\Sigma$.*

*Example 2.* Continuation of Example 1. $\langle 220 \rangle$ is a trustable answer to the query that asks for numbers of alarming samples.

Now that we have a satisfactory characterization of trustable answers, the remaining problem is to determine classes of queries and constraints for which trustable answers can be effectively computed. As announced in Section 1, we take the route of early-repairing.

**Definition 5.** *We say that a class* $\mathbf{Q}$ *of queries is* early-repairable *w.r.t. a class* $\mathbf{C}$ *of constraints iff for every satisfiable set* $\Sigma \in \mathcal{P}^{\text{fin}}(\mathbf{C})$*, for every relation* $I$*, there exists a computable relation* $I'$ *such that for every query* $Q \in \mathbf{Q}$*,* $Q(I')$ *is exactly the set of trustable answers to* $Q$ *on input* $I$ *and* $\Sigma$*.*

Hereafter, our focus will be on conjunctive queries and full dependencies.

## 6 Tableau Queries and Full Dependencies

We use the tableau formalism to express conjunctive queries. The following definition introduces the well-known constructs of tableau query and full dependency, which can be either tuple- or equality-generating. Unlike [6], tableaux need not be typed and can contain constants, and satisfaction is in terms of $\sim$.

**Definition 6.** *A* tableau query *is a pair* $(B, h)$ *where* $B$ *is a tableau and* $h$ *is a tuple (called* summary*) such that every variable in* $h$ *also occurs in* $B$*;* $B$ *and* $h$ *need not have the same arity. Let* $\tau = (B, h)$ *be a tableau query, and* $T$ *a tableau of the same arity as* $B$*. A tuple* $t$ *is an* answer *to* $\tau$ *on input* $T$ *iff there exists a substitution* $\theta$ *for the variables in* $B$ *such that* $\theta(B) \subseteq T$ *and* $\theta(h) = t$*. The set of all answers to* $\tau$ *on input* $T$ *is denoted* $\tau(T)$*.*

**Definition 7.** *A* full dependency *is either a* full tuple-generating dependency *(ftgd) or a* full equality-generating dependency *(fegd). An ftgd takes the form of a tableau query* $(B, h)$ *where* $B$ *and* $h$ *have the same arity. The ftgd* $\tau = (B, h)$ *is* satisfied *by a tableau* $T$*, denoted* $T \models \tau$*, iff* $T \cup \tau(T) \sim T$*. An fegd is of the form* $(B, p = q)$ *where* $B$ *is a tableau and* $p, q$ *are symbols such that every variable in* $\{p, q\}$ *also occurs in* $B$*. The fegd* $\epsilon = (B, p = q)$ *is* satisfied *by a tableau* $T$*, denoted* $T \models \epsilon$*, iff for every substitution* $\theta$*, if* $\theta(B) \subseteq T$ *then* $\theta(p), \theta(q)$ *are not two distinct constants and* $T \sim id_{\theta(p)=\theta(q)}(T)$*.*

*A tableau* $T$ *satisfies a set* $\Sigma$ *of full dependencies, denoted* $T \models \Sigma$*, iff for each* $\sigma \in \Sigma$*,* $T \models \sigma$*.*

*If* $E$ *is a tableau, a tableau query, an ftgd, or an fegd, then the* active symbol domain *of* $E$*, denoted* $\mathbf{asd}(E)$*, is the smallest set containing every symbol that occurs in* $E$*.*

*Example 3.* Consider a relation *Made* with four attributes, denoting date, product, color, and quantity respectively. The tuple $\langle 12\text{ Jan }2002, \text{lock}, \text{blue}, 110\rangle$, for example, means that 110 blue locks have been manufactured on 12 Jan 2002. The production line is subject to a number of constraints:

$\epsilon_1$

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| $x$ | $y$ | $z$ | $u$ |
| $x$ | $y$ | $z'$ | $u'$ |
| $z = z'$ | | | |

$\epsilon_2$

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| $x$ | $y$ | $z$ | $u$ |
| $x$ | $y$ | $z'$ | $u'$ |
| $u = u'$ | | | |

$\epsilon_3$

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 8 Jan 2002 | $y$ | $z$ | $u$ |
| $0 = 1$ | | | |

$\tau_1$

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| $x$ | lock | $z$ | $u$ |
| $x$ | key | $z$ | $u$ |

The fegd's $\epsilon_1$ and $\epsilon_2$ state that it is technically impossible to produce the same product in different colors on the same day, and that quantities are daily totals.

The fegd $\epsilon_3$ captures the fact that 8 Jan 2002 was a day of strike, on which no products were manufactured. Finally, the ftgd $\tau_1$ expresses that each production of a lock involves the simultaneous production of a key in the same color. Note, however, that it is possible to produce keys without locks.

A novelty, which is crucial for the properties to follow, is that satisfaction of full dependencies is expressed in terms of $\sim$. This implies that in verifying whether a tableau $T$ satisfies a full dependency, one must not simply treat $T$ as a relation by interpreting distinct variables as new distinct constants. For example, $T = \{\langle x, y\rangle, \langle z, z\rangle\}$ satisfies $\epsilon = (\{\langle x, y\rangle\}, x = y)$.

We now provide three lemmas that are needed in the main theorem to follow, which expresses that $\sim$-equivalent tableaux satisfy the same full dependencies.

**Lemma 3.** *Let $F$ be a tableau, and $\theta$ a substitution for the variables in $F$. Let $\tau = (B, h)$ be a tableau query. Then $\theta(\tau(F)) \subseteq \tau(\theta(F))$.*

**Lemma 4.** *Let $T, S$ be tableaux and $\tau$ a tableau query. If $T \succeq S$, then $\tau(T) \succeq \tau(S)$ and $T \cup \tau(T) \succeq S \cup \tau(S)$.*

**Lemma 5.** *Let $T, S$ be tableaux, and $p, q$ symbols. Let $\theta$ be a substitution such that $\theta(p), \theta(q)$ are not two distinct constants. If $\theta(S) \subseteq T$, then $id_{\theta(p)=\theta(q)}(T) \succeq id_{p=q}(S)$.*

**Theorem 1.** *Let $T, S$ be two tableaux and $\Sigma$ a set of full dependencies. If $T \sim S$ and $T \models \Sigma$, then $S \models \Sigma$.*

## 7 Greatest Lower Bounds

Every finite set of tableaux has a greatest lower bound under $\succeq$; the construction is borrowed from glb's of sets of clauses [15]. We show some interesting properties that suggest the use of glb's for early-repairing.

**Definition 8.** *Let* $\mathbf{S}$ *be a finite set of tableaux. The concept of lower bound is defined relative to the quasi-order $\succeq$. A tableau $L$ is a* lower bound *of* $\mathbf{S}$ *iff for each $T \in \mathbf{S}$, $T \succeq L$. A lower bound $G$ of* $\mathbf{S}$ *is called a* greatest lower bound *(glb) of* $\mathbf{S}$ *iff $G \succeq L$ for every lower bound $L$ of* $\mathbf{S}$.

*A* glb-mapping *is a partial one-one function $\mu : (\mathbf{dom} \cup \mathbf{var}) \times (\mathbf{dom} \cup \mathbf{var}) \stackrel{inj}{\rightarrow} \mathbf{dom} \cup \mathbf{var}$ such that $\mu(p, q) = p$ if $p = q$ is the same constant, and $\mu(p, q) \in \mathbf{var}$ otherwise. Let $T, S$ be tableaux and $\mu$ a glb-mapping whose domain includes $\mathbf{asd}(T) \times \mathbf{asd}(S)$. The glb-mapping $\mu$ is extended to pairs of tuples and pairs of tableaux as follows: if $t \in T, s \in S$, then $\mu(t, s) := \langle \mu(t(1), s(1)), \ldots, \mu(t(n), s(n))\rangle$; finally, $\mu(T, S) := \{\mu(t, s) \mid t \in T, s \in S\}$.*

It is easy to see that if some set $\mathbf{S}$ has a glb, then this glb is unique up to $\sim$.

**Theorem 2.** *Let $T, S$ be tableaux. If $\mu$ is a glb-mapping whose domain includes $\mathbf{asd}(T) \times \mathbf{asd}(S)$, then $\mu(T, S)$ is a glb of $\{T, S\}$.*

The following important theorem expresses that the constructs of glb and tableau query commute up to $\sim$.

**Theorem 3.** *Let $T, S$ be tableaux and $\tau = (B, h)$ a tableau query. Let $G$ be a glb of $\{T, S\}$, and let $F$ be a glb of $\{\tau(T), \tau(S)\}$. Then $\tau(G) \sim F$.*

**Corollary 1.** *Let $\mathbf{S}$ be a finite set of tableaux and $\tau = (B, h)$ a tableau query. Let $G$ be a glb of $\mathbf{S}$, and let $F$ be a glb of $\{\tau(T) \mid T \in \mathbf{S}\}$. Then $\tau(G) \sim F$.*

**Corollary 2.** *Let $\mathbf{S}$ be a finite set of relations and $\tau = (B, h)$ a tableau query. Let $G$ be a glb of $\mathbf{S}$, and let $J = \bigcap\{\tau(I) \mid I \in \mathbf{S}\}$. Then $\mathbf{grd}(\tau(G)) = J$.*

Finally, if a number of tableaux each satisfy some set of full dependencies, then so does their glb.

**Theorem 4.** *Let $T, S$ be tableaux and $\Sigma$ a set of full dependencies. Let $G$ be a glb of $\{T, S\}$. If $T \models \Sigma$ and $S \models \Sigma$, then $G \models \Sigma$.*

**Corollary 3.** *Let $\mathbf{S}$ be a finite set of tableaux and $\Sigma$ a set of full dependencies. Let $G$ be a glb of $\mathbf{S}$. If $T \models \Sigma$ for each $T \in \mathbf{S}$, then $G \models \Sigma$.*

## 8 Chasing Fixes

The chase was originally introduced as a tool for deciding logical implication; here we use it for repairing databases. We generalize some results of [6] to tableaux that can contain constants and need not be typed, replacing equality of tableaux by $\sim$. A consequence is that a chase can terminate "unsuccessfully" when it tries to identify two distinct constants. New results include decidability of subsatisfiability, which is needed for determining fixes.

**Definition 9.** *We introduce an artificial top element, denoted $\Box$, to the quasi-order $\langle \mathbf{T}, \succeq \rangle$. Let $T \neq \Box$ and $S$ be tableaux, and $\Sigma$ a set of full dependencies. We write $T \vdash_\Sigma S$ if $S$ can be obtained from $T$ by a single application of one of the following* chase rules*:*

1. *If $\tau = (B, h)$ is an ftgd of $\Sigma$, then $T \vdash_\Sigma T \cup \tau(T)$.*
2. *Let $(B, p = q)$ be an fegd of $\Sigma$, and $\theta$ a substitution such that $\theta(B) \subseteq T$. If $\theta(p)$ and $\theta(q)$ are two distinct constants, then $T \vdash_\Sigma \Box$; otherwise, $T \vdash_\Sigma id_{\theta(p)=\theta(q)}(T)$.*

*A* chase *of $T$ by $\Sigma$ is a maximal (w.r.t. length) sequence $T = T_0, T_1, \ldots, T_n$ of tableaux such that for every $i \in \{1, \ldots, n\}$, $T_{i-1} \vdash_\Sigma T_i$ and $T_i \neq T_{i-1}$.*

Requiring that chases be maximal tacitly assumes that chases are finite, which is expressed by Lemma 6.

**Lemma 6.** *Let $F \neq \Box$ be a tableau and $\Sigma$ a set of full dependencies.*

1. *If $T$ is a tableau in a chase of $F$ by $\Sigma$, then $T \succeq F$.*

2. *Each chase of $F$ by $\Sigma$ is finite.*
3. *If $T \neq \square$ is the last element of a chase of $F$ by $\Sigma$, then $T \models \Sigma$.*
4. *If $T \neq \square$ is the last element of a chase of $F$ by $\Sigma$, and $\theta$ is a valuation mapping distinct variables to distinct constants not occurring elsewhere, then $\theta(T) \models \Sigma$.*

The following two lemmas are needed for the main theorems to follow, expressing that the chase is Church-Rosser up to $\sim$ and that subsatisfiability is decidable for full dependencies.

**Lemma 7.** *Let $F, M$ be tableaux, both distinct from $\square$, and $\Sigma$ a set of full dependencies. Let $T \neq \square$ be a tableau in a chase of $F$ by $\Sigma$. If $M \succeq F$ and $M \models \Sigma$, then $M \succeq T$.*

**Lemma 8.** *Let $F, M$ be tableaux, both distinct from $\square$, and $\Sigma$ a set of full dependencies. If $\square$ is the last element of a chase of $F$ by $\Sigma$ and $M \succeq F$, then $M \not\models \Sigma$.*

**Theorem 5.** *Let $F \neq \square$ be a tableau and $\Sigma$ a set of full dependencies. If two chases of $F$ by $\Sigma$ end with $T$ and $S$ respectively, then $T \sim S$.*

Theorem 5 motivates the following definition.

**Definition 10.** *Let $F \neq \square$ be a tableau and $\Sigma$ a set of full dependencies. We write* $\mathbf{chase}(F, \Sigma)$ *for $[T]_\sim$ if the last element of a chase of $F$ by $\Sigma$ is $T$. The singleton $[\square]_\sim$ is also written $\square$.*

Given a set $\Sigma$ of full dependencies, it is decidable whether $\Sigma$ is satisfiable, and whether a given tableau $T$ subsatisfies $\Sigma$. The fegd $(\{\}, 0 = 1)$, for example, is unsatisfiable.

**Theorem 6.** *Let $\Sigma$ be a set of full dependencies and $F \neq \square$ a tableau. Then $F$ subsatisfies $\Sigma$ iff* $\mathbf{chase}(F, \Sigma) \neq \square$.

**Corollary 4.** *A set $\Sigma$ of full dependencies is satisfiable iff* $\mathbf{chase}(\{\}, \Sigma) \neq \square$.

Finally, all chases of tableaux that are equivalent under $\sim$, terminate with equivalent tableaux.

**Theorem 7.** *Let $T, S$ be tableaux, both distinct from $\square$, and $\Sigma$ a set of full dependencies. If $T \sim S$, then* $\mathbf{chase}(T, \Sigma) = \mathbf{chase}(S, \Sigma)$.

The results so far imply that we can effectively determine fixes for a relation and a set of full dependencies.

*Example 4.* The integrity constraints introduced in Example 3 are strongly violated by the relation *Made* shown in Fig. 2. The fixes $F_1$–$F_5$ are shown on top of their respective chase results. $F_1$ and $F_2$ assume that the date of 8 Jan 2002 and either color (red or blue) are mistaken. $F_3$ and $F_4$ assume that the date of 8 Jan 2002 and either product (key or lock) are mistaken. Finally, $F_5$ assumes that the date of 8 Jan 2002 should be replaced by different dates in either tuple of *Made*. It can be verified that any other fix is equivalent under $\sim$ to one of the five fixes shown.

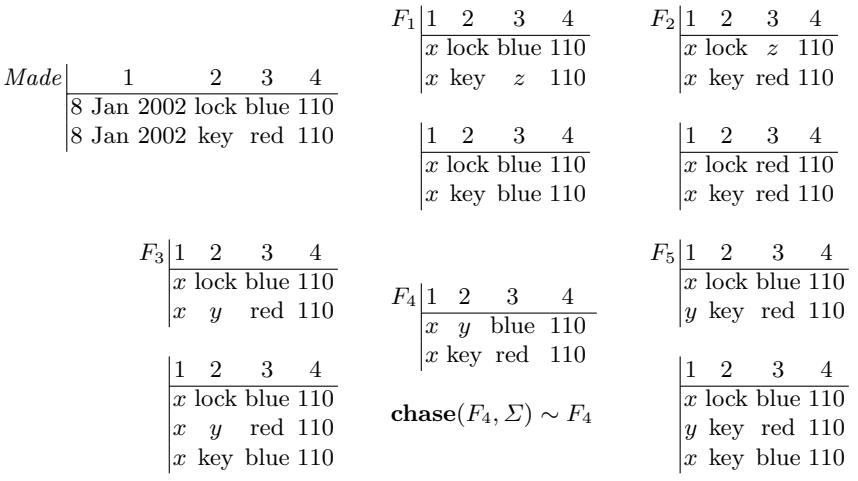



**Fig. 2.** Database with five fixes on top of chase results.

## 9 Condensed Representation

Our approach allows an infinite number of mends. In Fig. 2, mends are obtained by replacing in $F_1$–$F_5$ distinct variables by distinct constants not occurring elsewhere. According to Definition 4, the trustable answers to a query $Q$ on the relation *Made* are the tuples that are in the answer to the query on each possible mend. As there are infinitely many mends, this definition is impractical. We now develop an effective way for computing trustable answers.

**Definition 11.** *Let $I$ be a relation and $\Sigma$ a satisfiable set of full dependencies. Let $\mathbf{F}$ be a minimal (w.r.t. $\subseteq$) set of tableaux such that for every fix $F$ for $I$ and $\Sigma$, there exists some tableau $F' \in \mathbf{F}$ such that $F' \sim F$. Let $\mathbf{S}$ be a minimal (w.r.t. $\subseteq$) set of tableaux such that for every $F \in \mathbf{F}$, there exists some tableau $T \in \mathbf{S}$ such that $T \in \mathbf{chase}(F, \Sigma)$. Let $G$ be a glb of $\mathbf{S}$. Then $G$ is called a* trustable tableau *for $I$ and $\Sigma$.*

**Lemma 9.** *Let $I$ be a relation and $\Sigma$ a satisfiable set of full dependencies.*

1. *A trustable tableau for $I$ and $\Sigma$ exists and is computable.*
2. *If $G$ and $H$ are trustable tableaux for $I$ and $\Sigma$, then $G \sim H$.*

Lemma 9 motivates the following definition.

**Definition 12.** *Let $I$ be a relation and $\Sigma$ a satisfiable set of full dependencies. We write $\mathbf{trust}(I, \Sigma)$ for $[G]_\sim$ if $G$ is a trustable tableau for $I$ and $\Sigma$.*

The main result of the paper can now be stated.

**Theorem 8.** *Let $I$ be a relation and $\Sigma$ a satisfiable set of full dependencies. For every $G \in \mathbf{trust}(I, \Sigma)$,*

1. $G \models \Sigma$; *and*
2. *for every tableau query* $\tau$, $\mathbf{grd}(\tau(G))$ *is exactly the set of trustable answers to* $\tau$ *on input* $I$ *and* $\Sigma$.

Theorem 8 means that the class of tableau queries is "almost" early-repairable w.r.t. the class of full dependencies—almost, because there are some divergences from Definition 5. More precisely, for a given satisfiable set $\Sigma$ of full dependencies and relation $I$, compute a trustable tableau $G$ for $I$ and $\Sigma$. Satisfiability of $\Sigma$ can be determined by Corollary 4. Computability of $G$ follows from Lemma 9. From Theorem 8 it follows that for any tableau query $\tau$, $\mathbf{grd}(\tau(G))$ is exactly the set of trustable answers to $\tau$ on input $I$ and $\Sigma$; that $G$ also satisfies $\Sigma$, is then interesting but of less importance. The divergences from Definition 5 are the fact that $G$ is a tableau which can contain variables, and, related to that, the need to apply $\mathbf{grd}(\cdot)$ so as to restrict the query answer $\tau(G)$ to ground tuples. Intuitively, one can think of $G$ as a condensed representation of all mends that is sufficient for trustable query answering.

*Example 5.* Continuation of Example 4 and Fig. 2. A glb of all chase results is the trustable tableau $\{\langle x, \text{key}, z, 110\rangle\}$. It follows that $\langle \text{key}, 110\rangle$ is a trustable answer to the query $(\{\langle x, y, z, u\rangle\}, \langle y, u\rangle)$. The query $(\{\langle x, y, z, u\rangle\}, \langle y, z\rangle)$ has no trustable answers.

## 10 Size Issues

The results so far imply an effective way of constructing a trustable tableau. We now give some initial results on the efficiency of the construction, leaving a detailed complexity analysis for future work. Theorem 2 provides an upper bound on the minimal size of a glb of two tableaux. This bound is tight.

**Theorem 9.** *For all* $n, m \in \mathbb{N}$ *and* $n \geq 2$*, there exists tableaux* $T$ *and* $S$*, both of arity* $n$*, with* $|T|, |S| \geq m$ *such that no glb of* $\{T, S\}$ *has less than* $|T| * |S|$ *tuples.*

It should be noted, however, that in the construction of a trustable tableau, the glb is computed for a set of tableaux that result from fixing a common relation. We may hope that in practical situations, the database is "almost" consistent and the repair work only affects a limited number of tuples. In that case, all mends will have a large part in common. Such resemblances impact on the size of the glb.

**Theorem 10.** *If* $T, T_1, T_2$ *are tableaux such that* $\mathbf{asd}(T) \cap \mathbf{asd}(T_1) = \{\}$*, then there exists a glb of* $\{T \cup T_1, T \cup T_2\}$ *of size linear in* $|T|$.

What is the cardinality of a minimal trustable tableau for a relation $I$ and a set $\Sigma$ of full dependencies? We consider $\Sigma$ and the arity of $I$ fixed and look at the cardinality in terms of the size of $I$. Since the size of each fix is less than $|I|$, the chase of a fix results in a tableau whose size is polynomially bounded.

Example 6 shows that the number of non-equivalent (under $\sim$) fixes is not polynomially bounded. This does not preclude the existence of a trustable tableau of polynomial size, however. It is open whether the size of a trustable tableau is polynomially bounded in general.

*Example 6.* Let $V$ be a finite set of variables and $I = \{\langle a, a\rangle \mid a \in V\}$. Let $\epsilon = (\{\langle x, x\rangle\}, 0 = 1)$. Then every fix can be obtained from $I$ by putting in every row of $I$ a variable at the first or the second position, such that distinct rows use distinct variables. This yields $2^{|I|}$ non-equivalent fixes. The chase requires no further work. A trustable tableau for $I$ and $\{\epsilon\}$ is the singleton tableau $\{\langle x, y\rangle\}$.

## 11 Contributions and Related Work

Given a relation $I$ and a satisfiable set $\Sigma$ of integrity constraints, a ground tuple $t$ is a consistent (or trustable) answer to some query $Q$ on input $I$ and $\Sigma$ if $t$ is in the answer to $Q$ on each possible repair (or mend) for $I$ and $\Sigma$. The notion of repair can be formalized in different ways, and this gives rise to several notions of consistent query answer. Nevertheless, Corollary 2 implies that, whatever notion of repair is used, if the number of repairs for $I$ and $\Sigma$ is finite, then there exists a tableau $G$ such that for every tableau query $\tau$, $\mathbf{grd}(\tau(G))$ is exactly the set of trustable answers to $\tau$ on input $I$ and $\Sigma$. *Early-repairing* then means replacing $I$ by $G$ prior to the execution of tableau queries. It is easy to verify that if one adopts the repair construct of [3], which is tuple-based, then the number of repairs for any relation $I$ and any satisfiable set $\Sigma$ of full dependencies is finite, and consequently, conjunctive queries are early-repairable w.r.t. full dependencies (up to the application of $\mathbf{grd}(\cdot)$).

We argued, however, that tuple-based repairing is likely to be undesirable in practice, because tuples will be entirely deleted even if they contain only minor errors. It is not uncommon to have an operational database with large tuples, containing multiple errors in attributes that are of limited importance to the business. To deal with such situations, we developed value-based repairing, which allows remedying an error within a tuple without deleting the tuple. Using this finer repair construct, the number of possible repairs is no longer finite for full dependencies, and hence Corollary 2 does not apply in this case. The early-repairability of tableau queries w.r.t. full dependencies was established by Theorem 8: the construct of trustable tableau is a condensed representation of all (possibly infinitely many) mends that is sufficient for consistent query answering.

We left open bounds on the complexity of computing a trustable tableau $G$ for $I$ and $\Sigma$. We notice, however, that expensive repairs may be acceptable in practice if they do not occur frequently. Assume, for example, that $I$ and $\Sigma$ are part of a data warehouse that is updated only during an overnight batch run. After updating the data warehouse, a new trustable tableau $G$ for $I$ and $\Sigma$ has to be computed. The same trustable tableau can then be used throughout the day to compute trustable answers to any conjunctive query without additional repairing overhead, except for the removal of non-ground tuples. In these circumstances, early-repairing is preferable to late-repairing, which keeps the database

unchanged, but entails two overheads: first, each query needs to be transformed prior to its execution, and second, the transformed query will in general be much more expensive than the original one.

Although theoretical approaches to reasoning with inconsistent information date back to the 80s, the distinction between "consistent" and "inconsistent" answers to queries on databases that violate integrity constraints, seems to be due to Bry [8], who founded the idea on provability in minimal logic. Since then, several authors have searched to extend and adapt diverse existing formalisms so as to make them suited for consistent query answering, including analytic tableaux [7] (unrelated to our tableaux) and annotated predicate calculus [2]. Extended disjunctive logic programs with exceptions are used in [4] for extending the results of [3] to more general classes of queries and constraints. A further generalization is the technique proposed in [10,11], which is proved to compute exactly the consistent answers for universally quantified constraints; the language used is disjunctive Datalog with two forms of negation. [5] considers consistent query answering for aggregate queries and functional dependencies; several results rely on a compact (or condensed) graph-theoretical characterization of a relation relative to a set of functional dependencies, but unrelated to querying. Bertossi and Schwind [7] wonder whether compact representations of database repairs for consistent query answering can be attained after all. Our results show that such compact representations do exist for conjunctive queries and full dependencies. All approaches cited so far (including ours) assume a single database. The data integration setting of [9,12] considers a global database which is retrieved by queries over local source databases. Inconsistency can arise relative to integrity constraints expressed over the global schema. [12] studies the computation of consistent answers to queries over the global schema when the constraints are key constraints and foreign key constraints, the latter of which involve existential quantification. The works [2,3,4,5,7,11,12] all use a tuple-based notion of repair. To our knowledge, we are the first to consider the refined concept of value-based repairing.

Three problems that deserve future attention, are as follows. First, since a trustable tableau can be considered as a (complex) view, classical problems concerning views emerge. Of practical interest is the view maintenance problem: a trustable tableau has been computed and the problem is to maintain it incrementally when the original database is updated. Second, it is interesting to investigate whether our approach can be pushed to more expressive classes of queries. We already know that a single trustable tableau is not sufficient to compute all trustable answers if queries can be unions of conjunctive queries. Third, it is interesting to ask whether a route in between early- and late-repairing is advantageous, where one applies transformations to both the database and queries to obtain consistent query answers.

## References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

2. M. Arenas, L. Bertossi, and M. Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *Proc. 1st Int. Conf. on Computational Logic (CL 2000)*, volume 1861 of *LNAI*, pages 926–941. Springer, 2000.
3. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 68–79. ACM Press, 1999.
4. M. Arenas, L. E. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000)*, Advances in Soft Computing, pages 27–41. Springer, 2000.
5. M. Arenas, L. E. Bertossi, and J. Chomicki. Scalar aggregation in FD-inconsistent databases. In *Proc. 8th Int. Conf. on Database Theory (ICDT 2001)*, volume 1973 of *LNCS*, pages 39–53. Springer, 2001.
6. C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, Oct. 1984.
7. L. Bertossi and C. Schwind. Analytic tableaux and database repairs: Foundations. In *Proc. 2nd Int. Symposium on Foundations of Information and Knowledge Systems (FoIKS 2002)*, volume 2284 of *LNCS*, pages 32–48. Springer, 2002.
8. F. Bry. Query answering in information systems with integrity constraints. In *First IFIP WG 11.5 Working Conference on Integrity and Internal Control in Information Systems: Increasing the Confidence in Information Systems, Zurich, Switzerland, December 4-5, 1997*, pages 113–130. Chapman Hall, 1997.
9. A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *Proc. 14th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2002)*, volume 2348 of *LNCS*, pages 262–279. Springer, 2002.
10. G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. 17th Int. Conf. on Logic Programming (ICLP 2001)*, volume 2237 of *LNCS*, pages 348–364. Springer, 2001.
11. G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. on Knowledge and Data Engineering*, to appear.
12. D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*, number 54 in CEUR Workshop Proceedings, 2002.
13. J. Lin and A. O. Mendelzon. Merging databases under constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1998.
14. S.-H. Nienhuys-Cheng and R. de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. In L. D. Raedt, editor, *Advances in Inductive Logic Programming*, pages 265–276. IOS Press, 1996.
15. G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163, Edinburgh, 1969. Edinburgh University Press.
16. P. R. J. van der Laag and S.-H. Nienhuys-Cheng. Completeness and properness of refinement operators in inductive logic programming. *Journal of Logic Programming*, 34(3):201–225, 1998.

# Typing Graph-Manipulation Operations

Jan Hidders

University of Antwerp (UIA)
Universiteitsplein 1
B-2610 Antwerp, Belgium.
hidders@uia.ua.ac.be

**Abstract.** We present a graph-based data model called GDM where database instances and database schemas are described by certain types of labeled graphs called instance graphs and schema graphs. For this data model we introduce two graph-manipulation operations, an addition and a deletion, that are based on pattern matching and can be represented in a graphical way. For these operations it is investigated if they can be typed such that it is guaranteed for well-typed operations that the result belongs to a certain database schema graph, and what the complexity of deciding this well-typedness is.

## 1 Introduction

Since the introduction of the Entity-Relationship (ER) Model [1] labeled graphs have been used in many data models to represent schemas. This is not only the case for subsequent extensions of the ER Model but also in object-oriented data models such as LDM (the Logical Data Model) [2] and IFO [3].

Although the representation of instances as labeled graphs was already considered in FDM (the Functional Data Model) [4] the first data model that explicitly used labeled graphs as instances and regarded database transformation as graph transformations was GOOD (the Graph-Oriented Object Database Model) [5]. To represent complex values more naturally some data models have proposed to use generalizations of graphs such as *hypergraphs* [6], *hierarchical graphs* [7], *hygraphs* as used in the Hy$^+$ system [8], and finally graphs in the *hypernode model* [9].

The data model we propose here is based on the GOOD approach and represents both instances and schemas as labeled graphs called instance graph and schema graphs. The differences with GOOD are that (1) these two notions are defined independently such that instance graphs can exist without a corresponding schema graph and the data model can be used to represent *semistructured data* [10,11] and (2) the data model explicitly supports the notions of *complex values*, *inheritance* and $n$-ary *symmetric relationships* as are found in ER models.

The language introduced with the GOOD data model was one of the first graph-based languages that was shown to be able to express all *constructive* database transformations [12]. This was followed by languages such as PaMaL [13,14] and GOAL [15] that showed that only an addition and a deletion are

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 394–409, 2003.

sufficient to express all these transformation if certain nodes explicitly represent complex values. In this paper we show how these two operations can be redefined for our data model and we investigate the decidability of whether these operations respect a certain schema, i.e., whether the result of an operation will stay within that schema.

## 2 The Graph-Based Data Model GDM

### 2.1 Instance Graphs

In GDM an instance is represented by labeled graphs such as shown in Fig. 1 which are called *instance graphs*. The nodes in the graph represent entities, the edges represent attributes of these entities and the nodes are labeled with zero or more class names to indicate that they belong to certain classes. If from a certain node several edges leave that have the same label then this is interpreted as a single set-valued attribute. For example, the Department in Fig. 1 has an attribute sections that contains two sections. As is usual in object-oriented databases we distinguish three mutually exclusive kinds of entities [16]: *objects* which are represented by square nodes, *composite values* (or complex values) which are represented by round empty nodes and *basic values* which are represented by round nodes with a basic-type name inside and labeled with a representation of the basic value it represents.

Before we proceed with the formal definition of instance graphs we postulate the following symbols and sets. The postulated symbols are: **isa**, to label **isa** edges with, **is**, to label **is** edges with, **com**, to indicate composite-value nodes, and **obj**, to indicate object nodes. Given a set $X$ we let $\mathcal{P}(X)$ denote the *power set* of $X$, i.e., the set of subsets of $X$, and $\mathcal{P}_{fin}(X)$ denotes the set of *finite* subsets of $X$. The postulated sets are: the set $\mathcal{A}$ of attribute names, not containing **isa** or **is**, the set $\mathcal{B}$ of basic-type names, not containing **com** and **obj**, the set $\mathcal{C}$ of class names, the set $\mathcal{D}$ of representations of basic values, and the function $\delta : \mathcal{B} \to \mathcal{P}(\mathcal{D})$ that gives for every basic-type name the corresponding domain of basic-value representations.

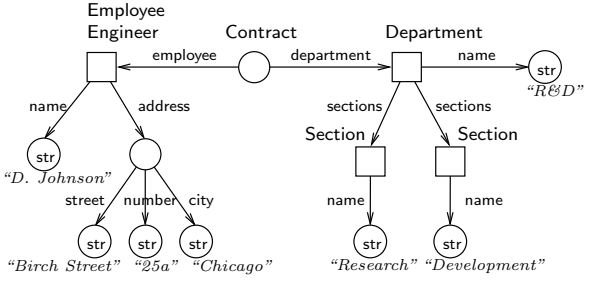



**Fig. 1.** An instance graph

The definition of instance graph will be based on the notion of *data graph*. A data graph is defined as $I = \langle N, E, \lambda, \sigma, \rho \rangle$ where $\langle N, E, \lambda \rangle$ is a finite labeled graph with node labels $\mathcal{P}_{fin}(\mathcal{C})$ and edge labels $\mathcal{A}$, and with the function $\sigma : N \to \{\mathbf{com}, \mathbf{obj}\} \cup \mathcal{B}$ that gives the *sort* of every node, and the function $\rho$ :

$\{ n \in N \mid \sigma(n) \in \mathcal{B} \} \rightarrow \mathcal{D}$ that gives a basic-value representation for basic-value nodes.

In a data graph nodes with sort **obj** are called *object nodes*, nodes with sort **com** are called *composite-value nodes* and nodes with a sort in $\mathcal{B}$ are called *basic-value nodes*. If $\lambda(n) = \emptyset$ then $n$ is called a *class-free node* and otherwise it is called a *class-labeled node*. If the components of $I$ are not explicitly named then they are presumed to be $N_I$, $E_I$, $\lambda_I$, $\sigma_I$ and $\rho_I$, respectively.

A data graph is called a *well-formed data graph* if (I-BVA) no edge leaves from a basic-value node, (I-BVT) if $\rho(n)$ is defined then $\rho(n) \in \delta(\sigma(n))$, (I-REA) for every class-free node $n$ there is a path that ends in $n$ and starts in a class-labeled node, and (I-NS) composite-value nodes have either exactly one incoming edge or are labeled with exactly one class name, but not both.

The constraints I-BVA and I-BVT follow from the meaning of basic-value nodes. The constraint I-REA is introduced to prevent "floating entities", i.e., entities that do not belong to any class and are not in the value of any attribute. Finally, the constraint I-NS is introduced because we assume that representations of composite values are weak entities that are not identified by only their attributes but also by the attribute or class that they belong to. This is consistent with, for example, the relational model and complex-value data models where an update to a tuple in a certain table or attribute does not imply that representations of the same tuple in other tables or attributes are also updated.

To establish when two nodes in a data graph $I$ represent the same complex value we define the *value-equivalence* relation $\cong_I \subseteq N_I \times N_I$ as the smallest reflexive relation such that (1) two basic-value nodes are value equivalent if they are labeled with the same basic-value representation and (2) two composite-value nodes $n_1$ and $n_2$ are value equivalent if for every edge $\langle n_1, \alpha, n_1' \rangle$ in $E_I$ there is an edge $\langle n_2, \alpha, n_2' \rangle$ in $E_I$ such that $n_1' \cong_I n_2'$ and vice versa.

Finally, we define *instance graphs* as well-formed data graphs for which it holds that (1) basic-value nodes are value-equivalent only if they are the same node, (2) two composite-value nodes that are labeled with the same class name are value-equivalent only if they are the same node and (3) two composite-value nodes that have both an incoming edge from the same node are value-equivalent only if they are the same node.

### 2.2 Schema Graphs

In GDM a schema is represented by labeled graphs such as shown in Fig. 2 which are called *schema graphs*. The nodes in a schema graph represent classes and the edges labeled with attribute names indicate that entities in that class may have that attribute. The nodes are labeled with zero or one class name to indicate the name of the class. As in data graphs the nodes have a sort which in this case indicates the sort of the entities in this class. The special edges that are not labeled with an attribute name but are drawn as hollow edges are **isa** edges that indicate that the class where the edge leaves is a subclass of the class where it arrives, i.e., all entities in the first class also belong to the second class.

Formally, a schema graph is defined as $S = \langle N, E, \lambda, \sigma \rangle$ where $\langle N, E, \lambda \rangle$ is a finite partially labeled graph, i.e., $\lambda$ may be undefined for some nodes, with node labels $\mathcal{C}$ and edge labels $\mathcal{A} \cup \{\mathbf{isa}\}$, and the function $\sigma : N \to \{\mathbf{com}, \mathbf{obj}\} \cup \mathcal{B}$ gives the *sort* of every node.

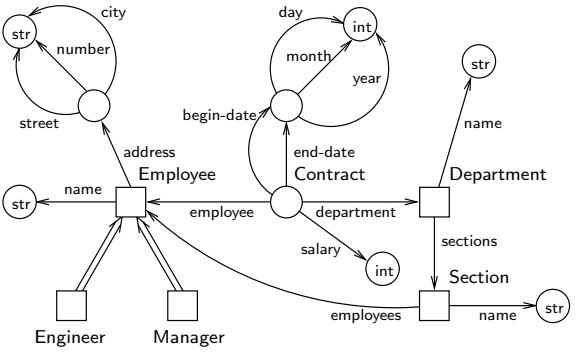



**Fig. 2.** A schema graph

Nodes with sort **obj** are called *object class nodes*, node with sort **com** are called *composite-value class nodes* and nodes with a sort in $\mathcal{B}$ are called *basic-value class nodes*. If $\lambda(n)$ is undefined then $n$ is called an *implied class node* and otherwise $n$ is called an *explicit class node*. The difference between an explicit class and an implied class is not that one has a name and the other does not, but rather that for the explicit class class-membership is explicitly indicated in the instance graph. Note that this is similar to the usual distinction between classes and types.

If the components of a schema graph $S$ are not explicitly named then they are presumed to be $N_S$, $E_S$, $\lambda_S$ and $\sigma_S$, respectively. The reflexive transitive closure of the binary relation over $N_S$ that is defined by the **isa** edges is written as $\mathbf{isa}_S^*$.

### 2.3 Extension Relations

The relationship between data graphs and schema graphs is established through *extension relations* which are many-to-many relations between the nodes in the data graph and nodes in the schema graph that indicate which entity belongs to which class. Such a relation must respect the meaning of the class names, the attribute edges and the **isa** edges, i.e., the extension relation must at least associate the nodes that should be associated according to these labels and edges. Therefore we define an extension relation from a schema graph $S$ to a data graph $I$ is defined as a relation $\xi \subseteq N_S \times N_I$ for which it holds that[1] (ER-CLN) if $\lambda_S(m)$ is defined and $\lambda_S(m) \in \lambda_I(n)$ then $\xi(m, n)$, (ER-ATT) if $\xi(m_1, n_1)$, $\langle n_1, \alpha, n_2 \rangle \in E_I$ and $\langle m_1, \alpha, m_2 \rangle \in E_S$ then $\xi(m_2, n_2)$, and (ER-ISA) if $\langle m_1, \mathbf{isa}, m_2 \rangle \in E_S$ and $\xi(m_1, n)$ then $\xi(m_2, n)$.

A node in the data graph should be labeled with a class name iff it belongs to a class with that name. Therefore we say that an extension relation $\xi$ from $S$ to $I$ is *class-name correct* if whenever $\lambda_S(m)$ is defined and $\xi(m, n)$ then $\lambda_S(m) \in \lambda_I(n)$.

[1] We will usually abbreviate $\langle m, n \rangle \in \xi$ to $\xi(m, n)$.

Another requirement is that nodes of different sorts may not be associated with each other. Therefore we say that an extension relation from $S$ to $I$ is *sort correct* if it only associates nodes with the same sort.

Finally, we require that all nodes, edges and class names in the data graph are somehow justified by corresponding nodes, edges and class names in the schema graph. We say that an extension relation $\xi$ from $S$ to $I$ *covers* $I$ if (CV-N) for every node $n \in N_I$ then there is some node $m \in N_S$ such that $\xi(m, n)$, (CV-E) for every edge $\langle n_1, \alpha, n_2 \rangle$ in $E_I$ there is some edge $\langle m_1, \alpha, m_2 \rangle$ in $E_S$ such that $\xi(m_1, n_1)$ and $\xi(m_2, n_2)$, and (CV-C) for every node $n \in N_I$ and class name $c \in \lambda_I(n)$ there is some explicit class node $m \in N_S$ such that $\xi(m, n)$ and $c = \lambda_S(m)$.

Summarizing, we say that a data graph $I$ *belongs to a schema graph* $S$ if it holds for the minimal extension relation from $S$ to $I$ that it is class-name correct, sort correct and covers $I$. It can be verified that under this definition the data graph in Fig. 1 belongs indeed to the schema graph in Fig. 2. To understand why only the *minimal* extension relation is considered look, for example, at the instance graph in Fig. 3 that would otherwise have belonged to the schema graph in Fig. 2 because the node at the end of the address edge might have been associated with the node at the end of the begin-date edge in the schema graph.

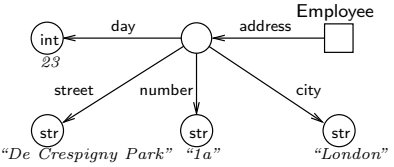



**Fig. 3.** An instance graph that not belongs to the schema graph in Fig. 2

## 2.4 Discussion of GDM

The purpose of GDM is to present a data model that is based on labeled graphs such that we can think of data manipulations as graph manipulations, and at the same time generalize over existing data models such as object-oriented data models and semistructured data models.

An example of a simulation of an object-oriented schema is shown in Fig. 4 (a). This schema graph defines a class B and a subclass A that inherits the attributes of B.

By allowing composite-value class nodes to play the same roles as object-class nodes GDM can also simulate the

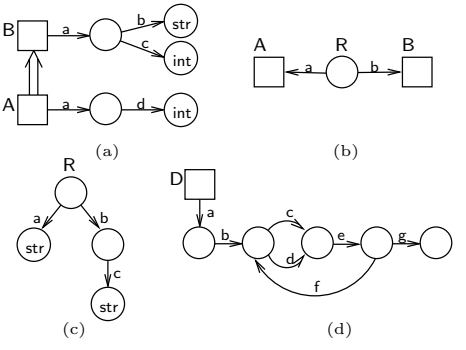



**Fig. 4.** Simulating other data models

relational model [17] and even the nested relational model [18] that allows non-first-normal-form relations and the Format Model [19] that allows arbitrary nesting of sets, tuples and tagged unions. An example of a simulation of this in GDM is given in Fig. 4 (c).

Moreover, since the attributes of composite values can also contain objects we can represent relationships between objects as shown in Fig. 4 (b). This enables GDM to simulate relationships as found in the ER model and ORM [20].

Finally, GDM can also be used to describe semistructured data because because instance graphs are self-describing and exist independently of any schema graph, it can represent arbitrarily nested values and a schema graph can express that the paths in a composite-value tree form a prefix of a certain regular expression. For example in Fig. 4 (d) the attribute of an D object can contain only paths that are a prefix of ab(c∪d)e(f(c∪d)e)*g.

## 3 The Graph-Based Update Language GUL

The basic mechanism of GUL is pattern matching. This means that every operation contains a pattern, i.e, a labeled graph that is similar to a well-formed data graph, and everywhere in the instance graph that this pattern can be embedded the operation is performed. Based on this mechanism we define an addition and a deletion and finally we introduce a reduction operation that reduces well-formed data graphs to instance graphs.

### 3.1 Patterns

The main difference between data graphs and patterns is the presence of **is** edges between composite-value nodes, which are drawn as hollow undirected edges. An **is** edge between two nodes specifies that these must be embedded upon two value-equivalent nodes in the instance graph. An example of a pattern with an **is** edge is given in Fig. 5. This pattern looks for an Engineer and a Manager that work for the same Section and live at the same address.

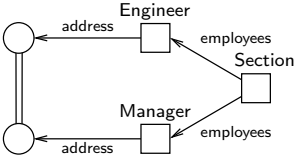



**Fig. 5.** A pattern with an **is** edge

More formally we define a pattern as a well-formed data graph $J = \langle N, E, \lambda, \sigma, \rho \rangle$ except that $\rho$ may be undefined for certain basic value nodes and we allow extra edges labeled with **is** between composite-value nodes if (1) these edges are symmetric, i.e., for every edge $\langle n_1, \mathbf{is}, n_2 \rangle$ there is an edge $\langle n_2, \mathbf{is}, n_1 \rangle$ and (2) in every cycle of edges with at least one attribute edge there is at least one object node. The final constraint is necessary because recursive composite-values are not allowed in well-formed data graphs.

An *embedding* of a pattern $J$ in a well-formed data graph $I$ is a function $h : N_J \to N_I$ that respects the class names, the sorts, the basic-value representations, the attribute edges and the **is** edges, i.e., for all nodes $n_1$ and $n_2$ in $N_J$

it holds that (1) $\lambda_J(n_1) \subseteq \lambda_I(h(n_1))$, (2) $\sigma_J(n_1) = \sigma_I(h(n_1))$, (3) if $\langle n_1, r\rangle \in \rho_J$ then $\langle h(n_1), r\rangle \in \rho_I$, (4) if $\langle n_1, \alpha, n_2\rangle$ in $E_J$ then $\langle h(n_1), \alpha, h(n_2)\rangle$ in $E_I$, and (5) if $\langle n_1, \mathbf{is}, n_2\rangle \in N_J$ then $h(n_1) \cong_I h(n_2)$. The set of all embeddings of $J$ into $I$ is written as $Emb(J, I)$.

### 3.2 The Addition

An addition is specified by giving two patterns; a *base pattern* $J$ and an *extension pattern* $J'$ that is, except for the **is** edges, a super-graph[2] of the base pattern with extra nodes, edges and class names such that all basic-value nodes in $J'$ that are not in $J$ are labeled with a basic-value representation. We will write such an addition as $\texttt{Add}(J, J')$ where $J$ is the base pattern and $J'$ is the extension pattern. An example of an addition is shown in Fig. 6.

The nodes, edges, and class names of the base pattern are drawn with normal lines and written in a normal font, whereas the additional nodes, edges and class names in the extension pattern are drawn with bold lines and written in a bold font.

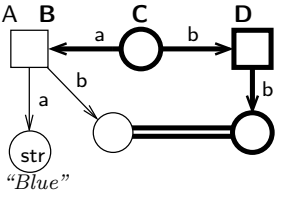



**Fig. 6.** An addition

Informally the result of applying an addition $\texttt{Add}(J, J')$ to a well-formed data graph $I$ is obtained by extending $I$ for each embedding of $J$ in $I$ with the nodes, edges and class names that are in $J'$ and it additionally adds nodes and edges to satisfy the **is** edges in $J'$. More formally we define the result of applying an addition $\texttt{Add}(J, J')$ to a well-formed data graph $I$, written as $[\![\texttt{Add}(J, J')]\!](I)$, as the well-formed data graph $I'$ where $I'$ is a minimal super-graph of $I$ such that there is a function $\eta : Emb(J, I) \to Emb(J', I')$ such that (1) $\eta(h)$ equals $h$ on $N_J$, (2) all distinct nodes in $N_{J'} - N_J$ are mapped by $\eta(h)$ to distinct nodes in $N_{I'} - N_I$, and (3) extensions of distinct embeddings map nodes in $N_{J'} - N_J$ to distinct nodes.

For example, the addition in Fig. 6 does for every embedding of the base pattern with the A node, the *"Blue"* node and the composite-value node the following: (1) it adds the extra nodes in the extension pattern, i.e., the C node, the D node and the second composite-value node, (2) it adds the class name B to the A node, and (3) it extends the new composite-value node such that it represents the same composite-value as the old composite-value node.

It is easy to see that the result of an addition without **is** edges in the extension pattern is always well-defined. However, in order for the result of an addition with **is** edges in the extension pattern to be always well-defined four well-formedness constraints are required. The first three are: (WA-CON) **is** edges are in $J'$ only allowed between nodes in $J$ and nodes not in $J$, (WA-NEC) if a node $n$ in $J'$ is not in $J$ and participates in an **is** edge in $J'$ then no attribute edge leaves from $n$, and (WA-NMC) every node in $J'$ that is not in $J$ is involved in at most one **is**

[2] The notion of *sub-graph* is defined as usual for labeled graphs except that nodes in a subgraph may be labeled with a subset of the corresponding set in the super-graph.

edge in $J'$. These constraints prevent that the addition has to merge composite values in order to satisfy **is** edges in $J'$. The reason that we want to prevent this is that there is not always a unique minimal extension of a weak instance graph that makes two composite-value nodes represent the same composite value.

The reason for the fourth and final well-formedness constraint is illustrated by the addition in Fig. 7. If we apply the addition (a) to the instance graph (b) as indicated by the nodes and edges drawn with solid lines, then we should extend it as indicated by the edges and nodes drawn with dotted lines, but this defines a composite value that contains itself which is not allowed.

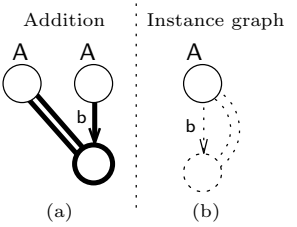



**Fig. 7.** An addition and its hypothetical result

To prevent such cycles we define the notion of *maximally merged version of an addition* $\texttt{Add}(J, J')$ which is constructed by merging the composite-value nodes and object nodes in $J$ if they have the same sort and the result is still an addition until no more nodes can be merged. The fourth well-formedness constraint then says that (WA-NRI) in the maximally merged version of the addition every cycle that contains at least one attribute edge also contains at least one object node.

Summarizing, an addition that satisfies WA-CON, WA-NEC, WA-NMC and WA-NRI is called a *well-formed* addition. It can be shown that the result of a well-formed addition is always well-defined since it can be constructed by first performing the addition without the **is** edges in the extension pattern, and then extending the data graph to satisfy the **is** edges in the extension pattern by copying for every **is** edge the sub-tree under the old node to the new node.

### 3.3 The Deletion

A *deletion* is specified by giving two patterns; a *base pattern* $J$ and a *core pattern* $J'$ that is, except for the **is** edges, a sub-graph of the base pattern. Both $J$ and $J'$ are patterns except that in $J'$ we allow class-free nodes that are not reachable from a class-labeled node. We will write such a deletion as $\texttt{Del}(J, J')$ where $J$ is the base pattern and $J'$ is the core pattern. An example of an addition is shown in Fig. 8.

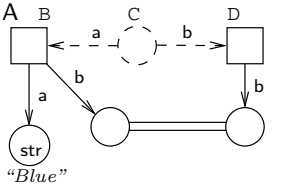



**Fig. 8.** A deletion

The nodes, edges, and class names of the core pattern are drawn with normal lines and written in a normal font, whereas the nodes, edges and class names that are not in the core pattern are drawn with dashed lines and written in an outline font.

The result of applying a deletion $\texttt{Del}(J, J')$ to a well-formed data graph $I$, written as $[\![\texttt{Del}(J, J')]\!](I)$, is defined as the well-formed data graph $I'$ where $I'$ is the maximal well-formed sub-graph of $I$ such that for every embedding $h$ of $J$ in $I$ it holds that (1) if $n \in N_J - N_{J'}$ then $h(n) \notin N_{J'}$, (2) if $\langle n_1, \alpha, n_2 \rangle \in E_J - E_{J'}$ then $\langle h(n_1), \alpha, h(n_2) \rangle \notin E_{J'}$, and (3) if for a node $n$ in $J$ there is a $c \in \lambda_J(n) - \lambda_{J'}(n)$ then $c \notin \lambda_{J'}(h(n))$.

Note that $I'$ can be constructed by first removing from $I$ for every embedding $h$ of $J$ in $I$ the nodes, edges and class names not in $J'$ and after that removing all class-free nodes that are no longer reachable from a class-labeled node.

### 3.4 The Reduction

The third and final operation of GUL is the *reduction* that transforms well-formed data graphs into instance graphs. It does this by merging two basic-value nodes if they are labeled with the same basic-value representation and merging two composite-value nodes if they are value equivalent and labeled with the same class name or both have an incoming edge from the same node, until no more nodes can be merged.

## 4 Typing GUL

### 4.1 Typing Patterns

A pattern $J$ is said to be *sound under a schema graph* $S$ if there is a well-formed data graph $I$ that belongs to $S$ and there is an embedding of $J$ in $I$. To detect such patterns we introduce a syntactical notion of well-typedness. The first case we consider is patterns with no **is** edges and schema graphs with no implicit object class nodes.

An extension relation $\xi$ from $S$ to $I$ is said to be *minimal on the composed-value nodes* if there is no strict subset of $\xi$ that is also an extension relation from $S$ to $I$ but is identical to $\xi$ on the object nodes and the basic-value nodes.

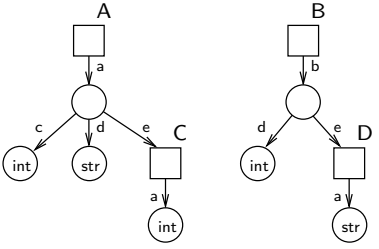



**Fig. 9.** A schema graph with no implicit object class nodes

**Definition 1.** *Given a schema graph $S$ with no implicit object class nodes a pattern $J$ without* **is** *edges is said to be* well-typed under *$S$ if there is an extension relation from $S$ to $J$ that* supports *$J$, i.e., that is minimal on the composed-value nodes and covers $J$.*

**Theorem 1.** *Given a schema graph $S$ with no implicit object class nodes a pattern $J$ without* **is** *edges is sound under $S$ iff it is well-typed under $S$.*[3]

[3] Proofs are omitted because of lack of space but are given in [21]

**Theorem 2.** *Deciding well-typedness of a pattern with no* **is** *edges under a schema graph with no implicit object class nodes is in PTIME.*

The next problem we consider is typing patterns with **is** edges. The introduction of **is** edges in patterns introduces extra typing problems. Consider, for example, the schema graph in Fig. 9 and the three patterns in Fig. 10. The problem in pattern (a) is that the **is** edge implies that there is a c edge under the b edge but this edge will not be covered in the schema graph in Fig. 9. The problem in pattern (b) is that the implied d edge ends in a node with the wrong sort. Finally, the problem in pattern (c) is that the implied e edge requires the C node to be in the class D and it should therefore be labeled with D, which may not be the case.

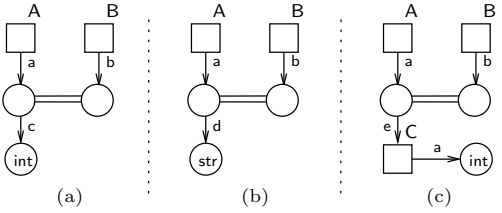



**Fig. 10.** Three patterns with **is** edges

To remedy this we have to check if these problems occur for all *value paths* in a pattern, where a value path is defined as a path of attribute, **is** and **isa** edges that contains only edges that start in composite-value nodes. If the list of attribute names that consecutively appear in such paths are the same then such paths are said to be *similar*. This leads to the following definition of well-typedness.

**Definition 2.** *Given a schema graph $S$ with no implied object classes a pattern $J$ is said to be* well-typed *under $S$ if there is an extension relation $\xi$ from $S$ to $J$ that* supports *$J$, i.e., it is minimal on the composite-value nodes, covers $J$ without the* **is** *edges and for every composite-value node $n$ in $J$ it holds for every value path in $J$ that starts in $n$ that (TP-CVV) if the path contains at least one attribute edge then there is a similar value path in $S$ starting in a node $m$ such that $\xi(m, n)$, (TP-CSV) for every similar value path in $S$ that starts in a node $m$ such that $\xi(m, n)$ it holds that these paths end in nodes with the same sort, and (TP-OCV) if the path ends in an object node $n'$ then it holds for every similar value path in $S$ that begins in $m$ such that $\xi(m, n)$ and ends in $m'$ that $\xi(m', n')$.*

**Theorem 3.** *Given a schema graph $S$ with no implicit object class nodes a pattern $J$ is sound under $S$ iff it is well-typed under $S$.*

**Theorem 4.** *Deciding well-typedness of a pattern under a schema graph with no implied object class nodes is co-NP complete.*

### 4.2 Typing Additions

A well-formed addition $\texttt{Add}(J, J')$ is said to *respect a schema graph $S$* if for every well-formed data graph $I$ that belongs to $S$ it holds that $[\![\texttt{Add}(J, J')]\!](I)$ also belongs to $S$.

The well typedness of a well-formed addition $\texttt{Add}(J, J')$ under a schema graph $S$ will be defined by considering all extension relations from $S$ to $J$ that support $J$ and then extending these minimally to $J'$. Obviously it should hold for this minimal extension that it covers $J'$ and is sort correct.

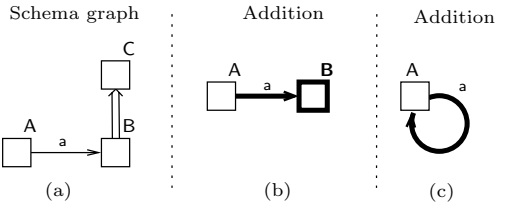



**Fig. 11.** A schema graph and two additions that add new nodes and edges

Two more constraints that should be checked are illustrated by Fig. 11. The addition (b) does not respect the schema graph (a) because the new node should also be labeled with C. So it should also be checked if the minimal extension places new nodes only in explicit classes that they are not labeled with. The addition (c) also not respects the schema graph for the same reason except here it is an old node that is placed in an extra explicit class that it is not already labeled with.

Two other constraints that should also be checked for the minimal extension of the supporting extension relation are illustrated by Fig. 12. The addition (b) does not respect the schema graph (a) because the A node might have an a edge ending in a int node. The int node would however conflict with the str class node at the end of the a edge from the B class node. The addition (c) also does not respect the schema graph (a) because here the A node might have an a edge ending in a class-free node. However, after the addition of the C label this node should be labeled with the D label, which it is not.

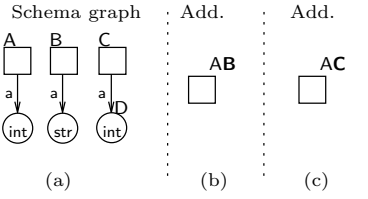



**Fig. 12.** A schema graph and two additions that add class names to old nodes

In order to prevent the previous two conflicts we introduce some new notions. A *weak value path* is a path of edges such that all inner nodes are composite-value nodes. Given a well-formed data graph $I$, a schema graph $S$ and an extension relation $\xi$ from $S$ to $I$ a *schema path for a noden* $\in N_I$ *under* $\xi$ is a path in $S$ that starts in a node $m$ such that $\xi(m, n)$. Such a path is said to be a *potential path* if for all prefixes $p'$ of the path there is not a path $p''$ from node $m'$ in $S$ such that $\xi(m', n)$ and $p'$ and $p''$ end in nodes with different sorts. It is now easy to see that the problems demonstrated in Fig. 12 can be detected by checking for all potential weak value paths if they cause sort problems or class-name problems at the ends of these paths.

**Definition 3.** *Given a schema graph $S$ with no implied object class nodes a well-formed addition* $\texttt{Add}(J, J')$ *with no* **is** *edges in $J'$ is said to be* well-typed under $S$ *if for every extension relation from $S$ to $J$ that supports $J$ the minimal superset $\xi'$ of $\xi$ that is an extension relation from $S$ to $J'$ is sort correct, covers $I_{J'}$ and (TA-*

*NCN) for every node $n$ in $J'$ that is not in $J$ it holds that if $\xi'(m,n)$ and $\lambda_S(m)$ is defined then $\lambda_S(m) \in \lambda_{J'}(n)$, and for every node $n$ in $J$ it holds that (TA-NCO) if $\xi'(m,n)$ and not $\xi(m,n)$ and $\lambda_S(m)$ is defined then $\lambda_S(m) \in \lambda_{J'}(n)$, (TA-CPW) every potential weak value path in $S$ for $n$ under $\xi$ is also a potential weak value path for $n$ under $\xi'$, and (TA-NPW) for every potential weak value path in $S$ for $n$ under $\xi$ and a similar path in $S$ for $n$ under $\xi'$ that ends in an explicit class node $m'_2$ there is a similar path in $S$ for $n$ under $\xi$ that ends in $m'_2$.*

**Theorem 5.** *Given a schema graph $S$ with no implied object class nodes a well-formed addition* Add$(J, J')$ *with no* **is** *edges in $J'$ respects $S$ if it is well-typed under $S$.*

Unfortunately it is not true the well-typedness is a necessary condition for respecting a schema graph as is shown in Fig. 13.

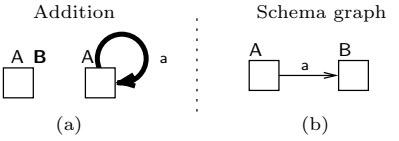



**Fig. 13.** A well-formed addition that respects a schema graph but is not well-typed

The presence of **is** edges in the extension pattern makes it necessary to check extra constraints. This is illustrated in Fig. 14. The addition (b) does not respect the schema graph (a) because the potential a attribute is copied to the new B node but according to the schema graph B nodes can only have b attributes. The addition (b) does not have this problem but here the sort of the potential a attribute under C is different from that of the a attribute under A. Finally the addition (c) does not respect (a) because the potential a attribute is forced into a new explicit class E with which it is not yet labeled.

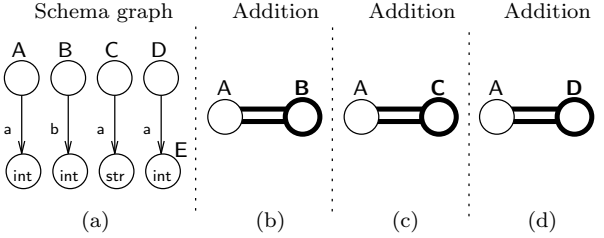



**Fig. 14.** A schema graph with three additions with **is** edges in the extension pattern

These considerations lead to the following extension of the definition of well-typedness.

**Definition 4.** *Given a schema graph $S$ with no implied object class nodes a well-formed addition* Add$(J, J')$ *is said to be* well-typed under *$S$ if it satisfies*

*the conditions for an addition without* **is** *edges in $J'$ and it holds for $\xi$ and $\xi'$ that for every* **is** *edge $\langle n_1, \mathbf{is}, n_2 \rangle$ in $J'$ with $n_1$ in $J$ it holds that (TA-CPI) for every potential weak value path in $S$ for $n_1$ under $\xi$ there is a similar potential weak value path in $S$ for $n_2$ under $\xi'$ that ends in a node with the same sort, and (TA-NPI) for every potential weak value path in $S$ for $n_1$ under $\xi$ and a similar path in $S$ for $n_2$ under $\xi'$ that ends in an explicit class node $m'_2$ there is a similar path in $S$ for $n_1$ under $\xi$ that ends in $m'_2$.*

**Theorem 6.** *Given a schema graph $S$ with no implied object class nodes a well-formed addition $\mathtt{Add}(J, J')$ respects $S$ if it is well-typed under $S$.*

**Theorem 7.** *Deciding well-typedness of a well-formed addition under a schema graph with no implied object class nodes is in PSPACE.*

**Theorem 8.** *Deciding if a well-formed addition respects a schema graph with no implied object class nodes is PSPACE hard.*

**Theorem 9.** *Deciding well-typedness of an addition under a schema graph with no implied object class nodes and no composite-value classes is in PTIME.*

### 4.3 Typing Deletions

Given a schema graph $S$ with no implied object class nodes a deletion $\mathtt{Del}(J, J')$ is said to *respect* $S$ if for all well-formed data graphs $I$ that belong to $S$ it holds that $[\![\mathtt{Add}(J, J')]\!](I)$ also belongs to $S$.

In order to understand what needs to be checked for well-typedness consider the deletion (a) in Fig. 15 and schema graph (b) in the same figure. In instance graph (a) the problem is that the class name that is deleted is required to be there since the node is still labeled with the name C of a subclass. In instance graph (b) the problem is that after the deletion the b edge is no longer covered by the schema graph. Finally, in instance graph (e) has the problem that after the deletion the c edge forces the node back into the B class.

To deal with these problems we have to consider two sets of class nodes: the set of nodes that an instance graph was associated with and the set of nodes that it is no longer associated with because the class names of these nodes are removed. For this purpose we introduce the notion of *basic deletion pair* which is defined for a deletion $\mathtt{Del}(J, J')$ and a schema graph $S$ with no implied object class nodes, as the set of pairs $\langle M_1, M_2 \rangle \in \mathcal{P}(N_S) \times \mathcal{P}(N_S)$ such that there is an extension relation $\bar{\xi}$ from $S$ to $J$ that supports $J$ and a node $n \in J$ such that $M_1 = \{\ m \in N_S \mid \bar{\xi}(m, n)\ \}$ and $M_2 = \{\ m \in N_S \mid \lambda_S(m) \in \lambda_J(n) - \lambda_{J'}(n)\ \}$.

Because embeddings are not injective and different embeddings can embed differently upon the same node, it is possible that what is removed from a node is a combination of basic deletion pairs. Therefore we define the set of deletion pairs as the smallest superset of the basic deletion pair that satisfies the rule

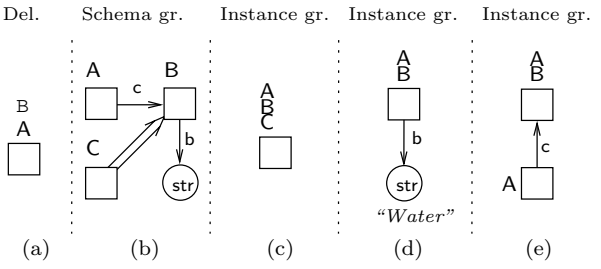



**Fig. 15.** A deletion, a schema graph and three instance graphs that demonstrate potential class name deletion problems

that if $\langle M_1, M_2 \rangle$ and $\langle M_1, N_3 \rangle$ are deletion pairs then $\langle M_1, M_2 \cup N_3 \rangle$ is also a deletion pair.

With the help of this set we can now define a notion of well-typedness that prevents the three problems that were demonstrated in Fig. 15.

**Definition 5.** *Given a schema graph $S$ with no implied object class nodes a deletion* $\texttt{Del}(J, J')$ *is said to be* well-typed under $S$ *if for every deletion pair $\langle M_1, M_2 \rangle$ with no composite-value class nodes in $M_1$ it holds that (TD-NSC) there are not two class nodes $m_1$ and $m_2$ in $S$ such that $m_1 \in M_1 - M_2$, $m_2 \in M_2$ and $m_1$ $\mathbf{isa}_S^*$ $m_2$, (TD-EC) there is not a potential weak value path $p$ in $S$ from $\{m_1\}$ where $m_1$ is an explicit class node $m_1$ and the end node $m_2$ of $p$ such that $m_2 \in M_2$ and for all similar weak value paths in $S$ from $m_1$ to $m_2'$ it holds that $m_2' \in M_1$, and (TD-CPE) for every potential weak value path in $S$ from $M_1$ there is similar weak value path in $S$ from $M_1 - M_2$.*

**Theorem 10.** *Given a schema graph $S$ with no implied object class nodes a deletion* $\texttt{Del}(J, J')$ *respects $S$ if it is well-typed under $S$.*

Unfortunately it is not true the well-typedness is a necessary condition for respecting a schema graph as is shown in Fig. 16.

**Theorem 11.** *Deciding well-typedness of an addition under a schema graph with no implied object class nodes is in PSPACE.*

**Theorem 12.** *Deciding if a deletion respects a schema graph with no implied object class nodes is PSPACE hard.*

### 4.4 Typing the Reduction

Typing the reduction is trivial because if a well-formed data graph belongs to a schema graph then the reduction will also belong to that schema graph. This can be understood if we look at one step of the reduction where two nodes are merged. It is easy to see that the result of merging these two nodes will belong to a schema graph iff the original well-formed instance graph belongs to it. It follows that the result of the reduction belongs to a schema graph iff the original well-formed instance graph does.

## 5 Conclusion

In this paper we introduced a graph-based data model GDM that represents instances and schemas as labeled graphs, and incorporates features from object-oriented data model, ER data models and semistructured data models. Together with this data model we introduced a graph-based update language GUL that is based on pattern matching and allows the user to specify additions and deletions in a graphical way.

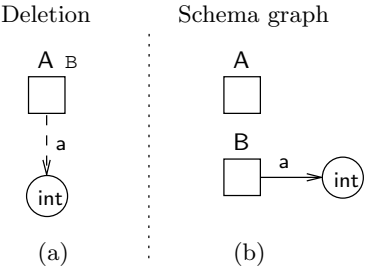



**Fig. 16.** A deletion that is not well-typed but respects the schema graph

For patterns a notion of well-typedness was introduced that captures exactly when there is a well-formed data graph that belongs to the schema graph and the pattern embeds into this data graph. For patterns without **is** edges this notion can be checked in PTIME and with **is** edges it was shown to be co-NP complete.

For additions a notion of well-typedness was introduced that is a sufficient condition, but not necessary condition, for an addition to respect a schema graph. Deciding this notion of well-typedness was shown to be PSPACE complete. In the special case where there are no composite-value nodes deciding well-typedness can be done in PTIME.

For the deletion also a notion of well-typedness was introduced that is a sufficient condition, but not a necessary condition, for a deletion to respect a schema graph. Deciding this notion of well-typedness was also shown to be PSPACE complete.

Finally the reduction operation that reduces all well-formed data graphs to instance graphs, was shown to be trivially well typed.

## References

1. Chen, P.P.: The Entity-Relationship Model: Toward a unified view of data. ACM Transactions on Database Systems **1** (1976) 9–36
2. Kuper, G.M., Vardi, M.Y.: The logical data model. ACM Transactions on Database Systems **18** (1993) 379–413
3. Abiteboul, S., Hull, R.: IFO: A formal semantic database model. ACM Transactions on Database Systems **12** (1987) 525–565
4. Shipman, D.W.: The functional data model and the data language DAPLEX. ACM Transactions on Database Systems **6** (1981) 140–173
5. Gyssens, M., Paredaens, J., Van den Bussche, J., Van Gucht, D.: A graph-oriented object database model. IEEE Transactions on Knowledge and Data Enginering **6** (1994) 572–586
6. Catarci, T., Tarantino, L.: A hypergraph-based framework for visual interaction with databases. Journal of Visual Languages and Computing **6** (1995) 135–166

7. Drewes, F., Hoffmann, B., Plump, D.: Hierarchical graph transformation. In: Foundations of Software Science and Computation Structure. (2000) 98–113
8. Consens, M.P., Eigler, F.C., Hasan, M.Z., Mendelzon, A.O., Noik, E.G., Ryman, A.G., Vista, D.: Architecture and applications of the $Hy^+$ visualization system. IBM Systems Journal **33** (1994) 458–476
9. Poulovassilis, A., Hild, S.G.: Hyperlog: a graph-based system for database browsing, querying and update. IEEE Data & Knowledge Engineering **13** (2001) 316–333
10. Abiteboul, S.: Querying semi-structured data. In: ICDT. (1997) 1–18
11. Suciu, D.: An overview of semistructured data. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory) **29** (1998)
12. Van den Bussche, J., Van Gucht, D., Andries, M., Gyssens, M.: On the completeness of object-creating database transformation languages. Journal of the ACM **44** (1997) 272–319 A revised and extended version of [22].
13. Gemis, M., Paredaens, J.: An object-oriented pattern matching language. In: Proceedings of the First JSSST International Symposium. Number 742 in LNCS, Springer-Verlag (1993) 339–355
14. Gemis, M.: Graph-based languages in DBMS. PhD thesis, University of Antwerp (1996)
15. Hidders, J., Paredaens, J.: GOAL: A graph-based object and association language. In Paredaens, J., Tenenbaum, L., eds.: Advances in Database Systems - Implementations and Applications. Volume 347 of CISM Courses and Lectures. Springer-Verlag (1994) 247–265
16. Beeri, C.: A formal approach to object-oriented databases. Data and Knowledge Engineering **5** (1990) 353–382
17. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM **13** (1970) 377–387
18. Jaeschke, G., Schek, H.J.: Remarks on the algebra of non first normal form relations. In: Proc. of the 1st ACM Symp. on Principles of Database Systems, Los Angeles, California (1982) 124–138
19. Hull, R.B., Yap, C.K.: The format model: A theory of database organization. Journal of the ACM **31** (1984) 518–537
20. Nijssen, G.M., Halpin, T.: Conceptual Schema and Relational Database Design: a fact oriented approach. Prentice Hall, Sydney, Australia (1989)
21. Hidders, J.: GUL, a Graph-based Update Language for Object-Oriented Data Models. PhD thesis, Eindhoven University of Technology (2001)
22. Van den Bussche, J., Van Gucht, D., Andries, M., Gyssens, M.: On the completeness of object-creating query languages. In: Proc. of the 33rd Symposium on Foundations of Computer Science, IEEE Computer Society Press (1992) 372–379

# Characterizing the Temporal and Semantic Coherency of Broadcast-Based Data Dissemination

Evaggelia Pitoura[1⋆], Panos K. Chrysanthis[2⋆⋆], and Krithi Ramamritham[3]

[1] Department of Computer Science, University of Ioannina, Greece
pitoura@cs.uoi.gr
[2] Department of Computer Science, University of Pittsburgh, USA
panos@cs.pitt.edu
[3] Department of Computer Science and Engineering, IIT, Bombay, India
krithi@cse.iitb.ac.in

**Abstract.** In this paper, we develop a general theory of temporal and semantic coherency for an extended client/server architecture in which the server broadcasts items of interest to a large number of clients without a specific client request. Such architectures are part of an increasing number of emerging applications in wireless mobile computing systems; they also provide a scalable means to deliver information in web-based applications, for example in publish-subscribe systems. We introduce various forms of temporal and semantic coherency applicable to such architectures and present a framework to precisely define protocols for enforcing them.

## 1 Introduction

While traditionally data is delivered from servers to clients on demand, a wide range of emerging data-based applications can benefit from a broadcast mode for data dissemination. In such applications, the server repetitively broadcasts data to a client population without explicit requests. Clients monitor the broadcast channel and retrieve the data items they need as they arrive on the broadcast channel. Such applications typically involve a small number of servers and a much larger number of clients with similar interests.

For example, in electronic commerce applications, such as auctions, it is expected that a typical auction might bring together millions of interested parties even though only a small fraction may actually offer bids. Updates based on the bids made must be disseminated promptly and consistently. Fortunately, the relatively small size of the database, i.e., the current state of the auction, makes broadcasting feasible. But, the communication bandwidth available for a client to communicate with servers is likely to be quite restricted. Thus, an attractive approach is to use the broadcast medium to transmit the current state of the

⋆ Work supported in part by IST-2001-32645
⋆⋆ Work supported in part by NSF award ANI-0123705

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 410–424, 2003.

auction while allowing the clients to communicate their updates (to the current state of the auction) using low bandwidth uplinks with the servers. Broadcast-based data dissemination is also likely to be a major mode of information transfer in mobile computing and wireless environments [9,1,6]. Many such systems have been proposed [17,11] and commercial systems such as Vitria [19] already support broadcasting. Other applications of broadcasting, include stock trading, next generation road traffic management systems and automated industrial plants [21].

**Motivation.** The problem addressed in this paper is motivated by such applications. In particular, we are concerned with providing readers with *consistent* (semantically coherent) and *current* (temporally coherent) data. By semantic coherency we mean the consistency properties of the data, e.g., *did the data items read by a client transaction result from a serializable execution of the update transactions at the server?* By temporal coherency we mean currency related properties, e.g., *when were the data items read by a client transaction current at the server?* Without both semantic and temporal coherency, users may be making decisions based on data which even if consistent, may be outdated. Given the limited amount of bandwidth available for clients to communicate with the broadcast server in such environments, achieving semantic and temporal coherency efficiently is a challenging research issue.

Several protocols have been proposed with the goal of achieving consistency and currency in broadcast environments. In [1], the authors discuss the tradeoffs between currency of data and performance issues when some of the broadcast data items are updated by processes running on the server. However, the updates do not have transactional semantics associated with them either at the server or at the clients. Herman et. al. [8] discuss transactional support in the Datacycle architecture, which is also an asymmetric bandwidth environment. Realizing that serializability as the correctness criterion may be expensive, and perhaps unnecessary in such environments, various protocols [16,2,13,12,10] attempt to cater to less demanding correctness requirements. However, the exact semantic and temporal coherency properties associated with them is not always clear. This lack of clarity has motivated the work reported in this paper. So, instead of proposing specific protocols for broadcast databases, as has heretofore been the case, we develop a unified framework for correctness in broadcast-based data dissemination environments.

**Overall Goals and Contributions.** Our framework allows us to characterize different semantic and temporal coherency properties of client transactions in broadcast-based data dissemination environments:

- We introduce the *currency interval* of an item in the readset of a transaction as the time interval during which the value of the item is valid. Based on the currency intervals of the items in the readset of a client transaction, we develop the notion of temporal spread and lag of the readset and two notions of currency (overlapping and oldest-value) through which we characterize the temporal coherency of the readset of a transaction.

– We identify five notions of semantic coherency of transactions' readsets. These are all variations or weakening of the standard serializability criterion.

Along the way we show what type of temporal and semantic coherency properties are guaranteed by the various protocols introduced in the literature and indicate how they can be extended to provide different properties.

**Paper Organization.** In Section 2, we introduce the broadcast model, give background definitions, and state the assumptions underlying our framework. In Section 3, after presenting our temporal coherency model, we present currency control protocols that satisfy different currency properties. In Section 4, we first define various models of semantic coherency that provide clients with transaction consistent database states and then present consistency control protocols based on the Read-Test theorem. We also present various propositions that relate certain types of temporal and semantic coherency. Finally, in Section 5, we present our conclusions.

## 2 The Model

We consider an extended client/server architecture in which a server broadcasts data items from a database to a large client population without a specific request from any client. Each client listens to the broadcast and fetches data items as they arrive. This way data can be accessed concurrently by any number of clients without the performance degradation that would result if the server were to transmit to individual clients. However, each client's access to the data is strictly sequential, since clients need to wait for the data of interest to appear on the channel. The broadcast may be periodic or aperiodic. Without loss of generality, we assume periodic broadcast.

### 2.1 Model Assumption

Data dissemination protocols take into account the asymmetry that exists between the communication capacity from the server to the clients, the typically meager communication capacity of the backchannel from the client to the server, and the scalability issues arising at the servers. The asymmetry is the result of the huge disparity in the transmission capabilities of clients and servers as well as the scale of information flow since a very large number of clients is connected to a single server. For scalability reasons, it is important to limit the number of explicit client requests seen by the server as well as individualized processing of each client request at the server. These reasons along with the need to decrease the latency of client transactions justify the use of client-side protocols for achieving client-specific semantic and temporal coherency requirements. These considerations motivate the following model assumptions:

1. The server is stateless: it does not maintain any client-specific information.

2. All updates are performed at the server and up-to-date data is disseminated from the server.
3. To get semantic and temporal coherency related information, clients do not contact the server directly instead such information is broadcast along with the data.

### 2.2 Update Period

Assume that a read operation on an item $x$ is initiated at time instance $t_r$. The client must wait for $x$ to appear on the channel, say, at time instance $t_x$. When we talk about the time of a read operation we refer to $t_x$ (i.e., the time instance the item is actually read) rather than to $t_r$. The value of $x$ that the client reads is the value placed on the channel at $t_x - d$, where $d$ is the communication delay.

Which value of an item $x$ is broadcast at a time instance depends on the update broadcast protocol. We consider a periodic update protocol with an *update cycle period* or *update frequency* $p_u$. With this protocol, the data values at the broadcast are updated every $p_u$ time units. Items broadcast during the same update period are said to belong to the same update cycle. We use the notation $begin\_cycle(t)$ for a time instance $t$ to denote the beginning of the update cycle period that includes $t$. The value of an item that the server puts on the broadcast at time $t_x$ is the value of $x$ at the server database at time instance $begin\_cycle(t_x)$ which may not be its current value at the database, if $x$ was updated between $begin\_cycle(t_x)$ and $t_x$. An update cycle period equal to 0, means that for each item the server broadcasts the most up-to-date value. In this case, $begin\_cycle(t) = t$.

Thus, the value that the client reads at $t_x$ from the broadcast is the value at the server database at $begin\_cycle(t_x - d)$. For simplicity, we will assume in the rest of this paper that the communication delay is negligible (i.e., $d = 0$).

In the following, we shall use the term cycle, to denote the update cycle. Usually, the update cycle period is considered equal to the period of the broadcast.

### 2.3 Preliminary Definitions

Next, we formally define the broadcast content and the transaction readset as subsets of a database state. A database state is typically defined as a mapping of every data item of the database to a value in its domain.

**Definition 1. (Database State)** *A databases state, denoted DS, is a set of (data item, value) pairs. The database state at time instance t is denoted as* $DS(t)$.

Let $BS_c$ be the set of (data item, value) pairs that appear on the broadcast during the cycle that starts at time instance $c$. $BS_c$ is a subset of a database state, in particular $BS_c \subseteq DS(c)$.

We use $R$ to denote a client read-only transaction. Let $t_{begin_R}$ and $t_{commit_R}$ be the time $R$ performs its first read operation and the time it commits, respectively.

**Definition 2. (Lifetime)** *The* lifetime *of R, denoted* lifetime(R) *is the time interval* $[t_{begin_R}, t_{commit_R}]$.

**Definition 3. (Readset)** *The readset of a transaction R, denoted RS(R), is the set of ordered pairs of data items and their values that R read. The readset includes at most one (data item, value) pair per data item. If a transaction reads the same item more than once, the readset includes the value read last. A readset is a subset of database state.*

In general, a transaction $R$ may read items from different cycles. In particular: $RS(R) \subseteq \cup_{c \in [t_1,t_2]} BS_c$, where $t_1 = begin\_cycle(t_{begin_R})$ and $t_2 = begin\_cycle(t_{commit_R})$. This says that the readset of $R$ is a subset of all the data items that are broadcast during the cycles that overlap with the execution of $R$.

## 3 Temporal Coherency

### 3.1 A Temporal Coherency Framework

We first define the currency of a single item read by a client.

**Definition 4. (Currency Interval of an Item)** *CI(x, R), the currency interval of x in the readset of R is* $[c_b, c_e)$ *where* $c_b$ *is the time instance the value of x read by R was stored in the database and* $c_e$ *is the time instance of the next change of this value in the database. If the value read by R has not been changed subsequently,* $c_e$ *is infinity.*

The above time instances ($c_b$ and $c_e$) correspond to *transaction* time [18]: the time that the value is stored in the database. They could as well correspond to *valid* time, i.e., the time when the value becomes effective in reality, or to some *user-defined* notion of time. The rest of this section holds for all such interpretations of time.

Based on the CIs of the items in the readset we define properties that characterize the currency of the readset. First, based on whether or not there is a time instance when the values read by $R$ are concurrently current at the server, we define two different forms of currency of the readset: *overlapping currency* and *oldest-value currency.*

**Definition 5. (Overlapping Currency of a Transaction)** *A transaction R is overlapping current if and only if there is an interval of time that is included in the currency interval of all items in R's readset:* $\cap_{(x,u) \in RS(R)} CI(x, R) \neq \emptyset$. *Let this overlap correspond to the interval* $[c_b, c_e)$, *then* $Overlapping\_Currency(R) = c_e^-$ *(where* $t^-$ *refers to the time instance just before t) if* $c_e$ *is not infinity, else* $Overlapping\_Currency(R) = current\_time$.

If a transaction $R$ is overlapping current and the intersection is the interval $[c_b, c_e)$, then its readset RS(R) is a subset of an actual database state $DS(t)$ at the server ($RS(R) \subseteq DS(t)$) for all $t \in [c_b, c_e)$.

If a transaction is not overlapping current, we characterize its currency based on the oldest value read by $R$ which is an indication of the outdatedness of the data.

**Definition 6. (Oldest Value Currency of a Transaction)** *The oldest-value currency of a transaction R, denoted OV_Currency(R), is equal to* $c_e^-$*, where* $c_e$ *is the smallest among the endpoints of the* $CI(x, R)$*, for every* $x$*,* $(x, u) \in RS(R)$*.*

For overlapping current transactions, oldest-value currency reduces to overlapping currency, that is, if $R$ is overlapping current then $OV\_currency(R) = Overlapping\_Currency(R)$.

In general, if a transaction is not overlapping current, its readset may not be a subset of a single database state $DS$ at the server. In this case, we are interested in minimizing the discrepancies in the currency of the items in the readset. This is captured through the notion of temporal spread.

**Definition 7. (Temporal Spread of a Readset)** *Let* $min_{c_e}$ *be the smallest among the endpoints and* $max_{c_b}$ *the largest among the begin-points of the currency intervals of items in the readset of a transaction R. The temporal spread of the readset of R, temporal_spread(R), is equal to (*$max_{c_b}$ *-* $min_{c_e}$*), if* $max_{c_b} > min_{c_e}$ *and zero otherwise.*

Temporal spread measures the discrepancies among the values read by a transaction; the larger the spread, the more distant in time the different server states seen by the transaction. For overlapping current transactions, the temporal spread is zero.

*Example 1.* (Fig. 1) Let data items $x_1$, $x_2$, $x_3$ and $x_4$ be in the readset of a transaction.

(i) Let $R_1$ be a transaction with $CI(x_1, R_1) = [2, \infty)$, $CI(x_2, R_1) = [4, 8)$, $CI(x_3, R_1) = [5, 10)$ and $CI(x_4, R_1) = [2, 18)$. Then, $R_1$ is overlapping current $Overlapping\_Currency(R_1)$=$8^-$ (the temporal spread is 0).

(ii) Let $R_2$ be another transaction with $CI(x_1, R_2) = [2, \infty)$, $CI(x_2, R_2) = [4, 8)$, $CI(x_3, R_2) = [5, 10)$ and $CI(x_4, R_2) = [9, 13)$. In this case, $R_2$ is not overlapping current, but is oldest-value current with $OV\_Currency(R_2) = 8^-$, meaning that the oldest value read by $R_2$ corresponds to $8^-$; all other values remain valid even after 8. Temporal_spread$(R_2) = 9-8 = 1$, that is the discrepancy between the database states seen by $R_2$ is 1.

(iii) Finally, let $R_3$ be a transaction $R_3$ with $CI(x_1, R_3) = [2, \infty)$, $CI(x_2, R_3) = [4, 10)$, $CI(x_3, R_3) = [5, 10)$ and $CI(x_4, R_3) = [15, 18)$. $OV\_Currency(R_3) = 9$ (i.e., $R_3$ reads less out-dated than $R_2$), however $R_3$'s temporal spread is equal to $15 - 10 = 5$, which is larger than the temporal spread of $R_2$.

We now examine how the currency of the readset relates to the transaction's lifetime. To this end, we define *transaction-relative currency* which relates the currency of the readset of a transaction $R$ with some time instance during its lifetime.

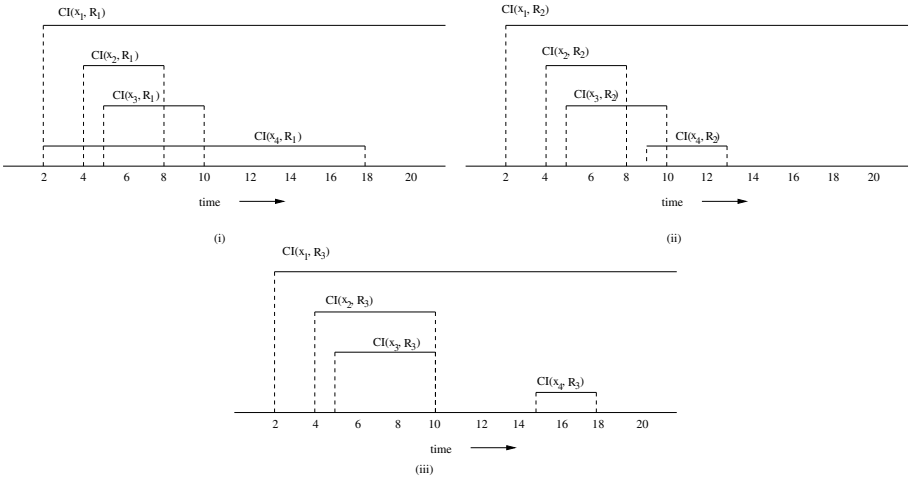



**Fig. 1.** Example 1

**Definition 8. (Transaction-Relative Currency)** *A transaction $R$ is* relative overlapping current *with respect to some time $t$, if $t \in CI(x, R)$ for all $x$ read by $R$. A transaction $R$ is* relative oldest-value current *with respect to some time $t$, if $t \leq OV\_Currency(R)$.*

For a given $R$, three possibilities of $t$ are: $t \geq t_{commit_R}$, $t_{begin_R} \leq t < t_{commit_R}$, and $t < t_{begin_R}$. Correspondingly, we may have $t$ to refer to the beginning of the respective cycle, that is, $t \geq begin\_cycle(t_{commit_R})$, $begin\_cycle(t_{begin_R}) \leq t < begin\_cycle(t_{commit_R})$, and $t < begin\_cycle(t_{begin_R})$.

**Definition 9. (Temporal Lag)** *Let $t_c$ be the largest $t \leq t_{commit_R}$, with respect to which $R$ is relative (overlapping or oldest-value) current, then $temporal\_lag(R) = t_{commit_R} - t_c$.*

Temporal lag indicates the currency of the values read; temporal lag of zero, means that the transaction reads items current at its commit time. The smaller the temporal lag and the temporal spread, the higher the *temporal coherency* of a read transaction. Thus, $R$ exhibits the best temporal coherency when it is overlapping relative current with respect to $t_{commit_R}$ (then both the time lag and the temporal spread are zero).

*Example 2.* (Fig. 2) If the lifetime of transaction $R_1$ in Example 1 is: (i) $[3, 7]$, then $R_1$ has temporal lag equal to 0, (ii) $[4, 12]$, then $R_1$ has temporal lag equal to 4 $(12 - 8)$, (iii) $[10, 19]$, then $R_1$ has temporal lag equal to 11 $(19 - 8)$. That is, although in all cases $R_1$ is overlapping current (i.e., the temporal spread is zero), $R_1$ in case (i) reads more current data than in case (ii) and this in turn is more current than in case (iii) (with respect to its lifetime).

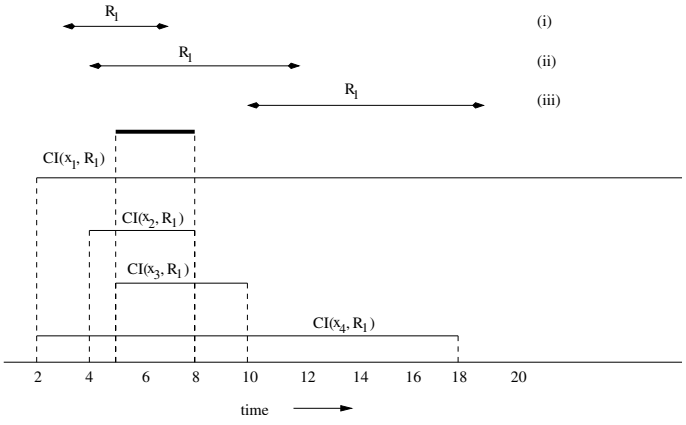



**Fig. 2.** Example 2

### 3.2 Achieving Temporal Coherency

Let us first see what kind of currency is attained when no additional control information is broadcast. The following proposition shows that we cannot guarantee anything better than oldest-value currency, in particular (proof in [14]):

**Proposition 1.** *Let $t_{lastread_R}$ be the time instance R performs its last read. If R reads items from the broadcast channel (without any additional information), then R is relative oldest-value current with respect to begin_cycle($t_{begin_R}$), with temporal_lag($R$) $\leq t_{commit_R} - begin\_cycle(t_{begin_R})$ and temporal_spread($R$) $\leq t_{lastread_R}$ - begin_cycle($t_{begin_R}$).*

From Proposition 1, we see that: the larger the period of the cycle, the better (smaller) the temporal spread and the worse (larger) the temporal lag. In fact, the best lag is attained when the period of the cycle is zero. In this case, $temporal\_lag(R) \leq t_{commit_R} - t_{begin_R}$ and $temporal\_spread(R) \leq t_{lastread_R}$ - $t_{begin_R}$.

If a transaction $R$ reads all items from the same cycle, then $temporal\_spread(R) = 0$, $R$ is overlapping current with respect to $begin\_cycle(t_{commit_R})$ and its $temporal\_lag(R)$ is equal to the update cycle period.

If we want to improve temporal coherency, additional information needs to be made available to clients. Next, we study some protocols that have appeared in the literature and formally express and prove their properties. When we say that a protocol satisfies a property, we mean that all transactions that follow the protocol have the property. The protocols fall in two broad categories: (a) invalidation (which corresponds to broadcasting the endpoints ($c_e$s) of the currency interval for each item) and (b) versioning (which corresponds to broadcasting the begin points ($c_b$s) of the currency interval for each item).

**Invalidation-based Protocols.** Invalidation is based on the following theorem (proof in [14]).

**Theorem 1. (Updates and Currency)** *Let $t_k$ be a time instance. Let $(x, u) \in RS(R)$ and $x$ be read at time instance $t_x$.*

*Condition (1) If $t_x \leq t_k$, then $x$ is not updated in (begin_cycle($t_x$), $t_k$].*
*Condition (2a) If $t_x > t_k$ and begin_cycle($t_k$) = begin_cycle($t_x$), then $x$ is not updated in (begin_cycle($t_x$), $t_k$].*
*Condition (2b) If $t_x > t_k$ and begin_cycle($t_x$) > begin_cycle($t_k$), then $x$ is not updated in ($t_k$, begin_cycle($t_x$)].*

*(A) Conditions (1) and (2a) hold for all $x$ in $RS(R)$ if and only if $R$ is relative oldest-value current with respect to $t_k$.*
*(B) Conditions (1), (2a) and (2b) hold for all $x$ in $RS(R)$ if and only if $R$ is relative overlapping current with respect to $t_k$.*

An invalidation report is a list with the items that have been updated at the server since the broadcast of the previous invalidation report. Without loss of generality, we assume that invalidation reports are broadcast periodically. This also facilitates selective tuning; a client can estimate the broadcast of the next invalidation report and tune in at the appropriate time. Let $p_i$ be the *invalidation period*; invalidation period equal to 0 means that an invalidation report for an item is broadcast immediately after the item is updated.

For notational convenience, we assume that broadcasting invalidation reports takes negligible time. Let $IR_c$ be the invalidation report that is sent at time instance $c$, $IR_c$ includes all items that have been updated since the broadcast of the invalidation report $IR_{c-p_i}$.

For a time instance $t$, we use the notation $invalidation(t)$ to denote the time instance when the invalidation report covering $t$ is broadcast, that is the time instance $t' \geq t$ when the next invalidation report is broadcast. That is, for $c - p_i < t \leq c$, $invalidation(t) = c$. For $p_i = 0$, $invalidation(t) = t$. In the case in which the period of broadcasting invalidation reports ($p_i$) is equal to the update period ($p_u$), invalidation reports are broadcast at the beginning of each cycle, and $IR_{begin\_cycle(t)}$ includes all items that have been updated since the beginning of the previous cycle, and covers all $t$, $begin\_cycle(t) - p_u < t \leq begin\_cycle(t)$. In particular, $begin\_cycle(t)$ is covered by $IR_{begin\_cycle(t)}$ (that is, $invalidation(begin\_cycle(t)) = begin\_cycle(t)$).

The following property relates invalidation reports and updates (proof in [14]).

*Property 1 (Invalidation).* For a data item $x$, let two time instances $t_1$ and $t_2$, $t_2 > t_1$, $x$ is not updated in the time interval $[t_1, invalidation(t_2)]$ iff $x \notin IR_c$, for every $c$, $invalidation(t_1) \leq c \leq invalidation(t_2)$.

Using Property 1 and Theorem 1, we can design various invalidation protocols and prove their currency properties. In particular, to guarantee that a transaction $R$ is oldest-value (overlapping) current with respect to some time instance, we need to test for each item in its readset whether it is updated in the specific interval defined in Theorem 1. To do so, we read the corresponding

invalidation reports specified by Property 1. In the case that the item appears in any such report, the transaction is aborted.

Note that, using Property 1, guarantees oldest-value (overlapping) currency with respect to $invalidation(t_k)$ as opposed to $t_k$. In other words, the finest granularity that we can get through invalidation reports is at the time the invalidation report is sent; in other words, we can only see the database states $DS(t)$ that correspond to invalidation points. In the special case in which $p_i = p_u$ (that is when invalidation reports are broadcast at the beginning of each cycle), we get currency with respect to *begin_point*s.

If invalidation reports are broadcast more often (small $p_i$), then we need to read more reports. However, we need to listen to the broadcast for less time, since the last report to be read is the report that covers $t_2$ and how long after $t_2$ this report will be broadcast depends on $p_i$.

A specific instance of the invalidation protocol with $t_k = begin\_cycle(t_{begin_R})$ appears in the literature as invalidation lists [13,12] or certification reports [2] (if we consider only read-only transactions).

A variation of invalidation report is based on re-reading an item that appears in an invalidation report, instead of aborting the issuing transaction. Such a method called *autoprefetch* was proposed in [1]. There, it was used to achieve overlapping currency with respect to $begin\_cycle(t_{commit_R})$.

**Versioning.** Another basic approach for ensuring overlapping currency is based on versions or timestamps [12]. In particular, with each (data item, value) pair in the broadcast, $(x, u) \in BC_c$, we also broadcast a version number or timestamp, $timestamp(x) = c_b$, where $c_b$ is the time instance the value of $x$ was written in the database. Let $t_0 = begin\_cycle(t_{begin_R})$. When $R$ reads $x$, if $timestamp(x) > t_0$, $R$ is aborted. It is easy to show (proof in [14]) that

*Claim.* The versioning protocol ensures that a transaction $R$ is overlapping current with respect to $t_0$, $t_0 = begin\_cycle(t_{begin_R})$.

The following protocol [12] uses both invalidation reports and versioning. Let $IR_{t_i}$ be the first invalidation report that includes an item previously read by $R$. Until $t_i$, $R$ reads items as they appear in the broadcast. After $t_i$, $R$ checks whether $timestamp(x) > t_i$, if so, $R$ is aborted. It is easy to show (proof in [14]) that

*Claim.* The versioning protocol with invalidation reports ensures that a transaction $R$ is overlapping current with respect to $t_i$, where $IR_{t_i}$ is the first invalidation report that includes an item previously read by $R$.

## 4 Semantic Coherency

### 4.1 A Semantic Coherency Framework

The currency properties of a client transaction focus on the timeliness of its readset. In this section, we consider whether the data read by a transaction are semantically correct. We relate semantic coherency with database consistency.

**Definition 10. (Database State and its Consistency)** *A* database state is consistent *if it does not violate the integrity constraints [3] defined on the database. A* subset of a database state is consistent *if it is a subset of a consistent database state [15].*

Thus, since a readset of a transaction is a subset of a database state, it is consistent if it is a subset of a consistent database state.

A consistent database state need not necessarily be produced by a set of server transactions. Hence, various consistency guarantees stronger than correspondence to a consistent database state have been defined, based on transaction serializability [4,7,20]. All (except one) guarantee that a read-only client transaction sees a consistent database state. We discuss them now in increasing order of "strength":

**Definition 11. ( Degrees of Consistency)** *Let R be a read-only transaction, then*

*R is* **C0** *consistent if no consistency requirements are placed on its readset.* No consistency[1] *[7] and* opportunistic *consistency [1] are examples of occurrence of C0 in the literature.*
*R is* **C1** *consistent if RS(R) is a subset of a consistent database state. No other serializability-based requirements are placed on this state.* Consistency *[20] and* weak consistency *[7] are examples of occurrence of C1 in the literature.*
*R is* **C2** *consistent if R is serializable with the set of server transactions that produced values that are seen (either directly or indirectly) by R.* Update consistency *[4,16],* weak consistency *[5] and* external consistency *[20] are examples of C2 from the literature.*
*R is* **C3** *consistent if R is serializable with the set of* all *server transactions.* Weak consistency *[4] is an example of C3 from the literature.*
*R is* **C4** *consistent if R is serializable with the set of all server transactions and observes a serial order of server transactions that agrees with the order in which the server transactions committed.*

A schedule is *rigorous* iff the commit order of transactions is compatible with the serialization order. So, C4 is C3 plus the requirement that schedules be rigorous.

Each criterion in the above list strengthens the previous criterion. Let us examine this list working backwards: C4 demands the serializability of all server transactions and *R*. In addition, it demands that the serialization order be consistent with the commit order. C3 is derived by dropping from C4 the requirement dealing with the serialization order having to agree with the commit order. So,

[1] We have retained the original terms, even though some terms like "strong" and "weak" have been used by different authors to mean different things. By formulating a framework for discussing correctness requirements, our hope is to shed some light on their precise meanings.

C3 simply demands the serializability of all server transactions and $R$. C2 is derived from C3 by demanding the serializability of a read-only transaction $R$ with just those server transactions from which $R$ reads data directly or indirectly. C1 drops the requirement of serializability from C2, being satisfied with the readset simply being consistent. C0 drops even the consistency requirement from C1.

### 4.2 Relation to Temporal Coherency

In the following, we adapt the definition of the currency interval to reflect that the server broadcast items from a transactional database system. The temporal coherency framework holds for this adapted definition as well.

**Definition 12. ((Transactional) Currency Interval of an Item)** *CI(x, R), the currency interval of $x$ in the readset of $R$ is $[c_b, c_e)$ where $c_b$ is the commit time of the transaction that wrote the value of $x$ read by $R$ and $c_e$ is the commit time of the transaction that next changes this value. If the value read by $R$ has not been changed subsequently, $c_e$ is infinity.*

Our only assumption about concurrency control at the server is that server schedules are serializable and that only committed values are broadcast. It can be proved ([14]) that:

**Proposition 2.** *If a client transaction $R$ is overlapping current, then $R$ is C1 consistent.*

Overlapping current transactions using the transactional definition of the currency interval correspond to the $c_e$-*vintage* transactions of [7]. A transaction satisfies the $t$-vintage requirement iff it reflects the updates of all transactions committed before time instance $t$ and does not reflect any updates due to any transaction committed after $t$. Similarly, a concept related to oldest-value current transactions are $t$-*bound* transactions [7]. A $t$-bound transaction sees the results of all transactions committed prior to time instance $t$ but also of some transactions committed after $t$. Thus, we may say that a transaction $R$ with $OV\_Currency(R) = t_o$, is $t_o$-bound.

### 4.3 Achieving Semantic Coherency

Proposition 2 shows that to attain C1 consistency, it suffices to attain overlapping currency. However overlapping currency is not sufficient to attain stricter notions of consistency even when a transaction reads all items from the same cycle (proof in [14]):

**Proposition 3.** *If a client transaction reads all items from a single cycle, it is C1, but not necessarily C2.*

**The Read-Test.** As the following theorem shows, in order to efficiently check for serializability-based consistency (that is, C2, C3, or C4 consistency), the server schedules should be rigorous. The theorem states (proof in [14]) that when server schedules are rigorous, checks of consistency violations can be done when a new item is read by a client transaction. This simplifies the construction of protocols for attaining semantic coherency.

**Theorem 2. (Read-Test Theorem)** *It suffices to check for C2, C3, or C4 consistency violations when a read-only transaction reads a new data item if and only if the schedule of server transactions is rigorous.*

The proof of the Read-Test theorem also shows that if the server schedule is not rigorous, then any of the C4, C3, and C2 consistency may be violated anytime a server transaction $T_f$ writes an item that was previously read by a client transaction $R$ and there is some path from $T_f$ to a server transaction $T$ from which $R$ read an item. This can happen even after $R$ has completed its operations.

Note that even when the server schedules are rigorous, C3 is not equivalent to C4. For example, let $T_0$, $T_1$, $T_2$ be server transactions and $T_R$ be a client transaction. Consider the schedule $w_0(y)\ c_0\ r_R(y)\ ||\ w_1(y)w_2(x)c_1c_2\ ||\ r_R(x)c_R$, where $||$ denotes the beginning of a new cycle. The commit order (which is compatible with the serializability) order at the server is: $T_0 \rightarrow T_1 \rightarrow T_2$. The serializability order at the client it is: $T_0 \rightarrow T_2 \rightarrow T_R \rightarrow T_1$, which means that $T_R$ is C3 but not C4.

From Theorem 2, we get that:

**Corollary 1.** *If the server schedule is rigorous and $R$ reads all items from the same cycle, then $R$ is C4.*

Thus, there is a trade-off between temporal and semantic coherency. The larger the length of the cycle (i.e., the update frequency) the worst the temporal spread, but the stronger the consistency attained.

**Semantic Coherency Protocols.** Let us consider what is needed to achieve C2, C3 and C4 consistency, in the case of rigorous schedules (proofs in [14]).

**Corollary 2.** *Let the server schedule be rigorous, $R$ read $x$ from $T$, and $Follow_R = \{T_f$: $T_f$ overwrote an item previously read by $R\}$. Then,*

*C4 Read Test* *Let $t_{min} = min_{T_f \in Follow\_R}\ (t_{commit_{T_f}})$ and $t_T = t_{commit_T}$, $R$ is C4 iff $t_T < t_{min}$.*

*C3 Read Test* *$R$ is C3 iff there is no path from any $T_f \in Follow_R$ to $T$ in the serialization graph of server transactions.*

*C2 Read Test* *$R$ is C2 iff there is no path from any $T_f$ to $T$ that includes only dependency transaction edges in the serialization graph of server transactions. Dependent transaction edges are those edges that correspond to transactions that $R$ "directly or indirectly reads from".*

**Proposition 4.** *If a transaction R is C4 consistent, then R is overlapping current with respect to $t_{min}$ of Corollary 2.*

The BCC-T1 method [10] provides an implementation of the C4 test. The commit timestamp of the transaction that last wrote each item is also broadcast along with the item. In addition, an invalidation report is broadcast periodically that includes for each data item $x$ that has been updated since the previous report the pair ($x$, $min\_t$) where $min\_t$ is the smallest commit timestamp among all transactions that wrote $x$. For each transaction R, we also maintain the set $Current\_RS(R)$ that includes the (item, value) pairs read by R so far and a counter *count* as follows. When for an item $(x, value) \in Current\_RS(R)$ the pair ($x$, $min\_t$) appears in an invalidation report, we set *count* equal to min{$min\_t$, count}. For each item read, the Read-Test checks whether the item read has timestamp less than *count*. If this does not hold, $R$ is aborted.

The SGT method [13] provides an implementation of the C3 test. The server maintains a serialization graph SG with all transactions committed at the server. The server broadcasts periodically the serialization graph to the clients. Each clients maintains a local copy of the graph. The Read-Test checks for cycles in the local copy of the graph.

The F-matrix [16] provides an interesting implementation of the C2 test. Along with each item x, an array C with $n$ elements (where $n$ is the number of items in the database) is broadcast. C[i] provides information regarding the transaction that affected the values of both items $x$ and $i$.

## 5 Conclusions

In this paper, we have proposed a general theory for characterizing the temporal and semantic coherency of transactions for an extended client/server environment in which the server broadcasts items of interest to a large number of clients. Our model provides the necessary tools for arguing about the correctness and other properties of the various protocols. In addition, it provides the basis for new protocols to be advanced. The proposed model can be easily extended for the case of a cache being maintained at the clients. In this case, clients read items from the broadcast channel or from the cache. The theory is directly applicable to caches. if the values in cache are maintained current. It can also be extended for deferred cache update policies.

## References

1. S. Acharya, M. J. Franklin, and S. Zdonik. Disseminating Updates on Broadcast Disks. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB 96)*, September 1996.
2. D. Barbará. Certification Reports: Supporting Transactions in Wireless Systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 1997.

3. P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addisson-Wesley, 1987.
4. P. M. Bober and M. J. Carey. Multiversion Query Locking. In *Proceedings of the 1992 SIGMOD Conference*, pages 497–510, 1992.
5. A. Chan and R. Gray. Implementing Distributed Read-Only Transactions. *IEEE Transactions on Software Engineering*, 11(2):205–212, 1985.
6. A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM TODS*, 24(1), 1999.
7. H. Garcia-Molina and G. Wiederhold. Read-Only Transactions in a Distributed Database. *ACM TODS*, 7(2):209–234, 1982.
8. G. Herman, G. Gopal, K.C. Lee, A. Weinreb, "The Datacycle Architecture for Very High Throughput Database Systems," Proceedings of the ACM SIGMOD Conference, New York, 1987.
9. T. Imielinski, S. Viswanathan, and B. R. Badrinanth. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, May/June 1997.
10. V. C. S. Lee, S. H. Son, and K. Lam. On the Performance of Transaction Processing in Broadcast Environments. In *Proccedings of the International Conference on Mobile Data Access (MDA'99)*, 1999.
11. B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus – An Architecture for Extensible Distributed Systems," Proceedings of the SOSP Conference, North Carolina, December 1993.
12. E. Pitoura and P. K. Chrysanthis. Exploiting Versions for Handling Updates in Broadcast Disks. In *Proceedings of 25th VLDB*, pages 114–125, 1999.
13. E. Pitoura and P. K. Chrysanthis. Scalable Processing of Read-Only Transactions in Broadcast Push. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999.
14. E. Pitoura, P. K. Chrysanthis, and K. Ramamritham. Characterizing the Semantic and Temporal Coherency of Broadcast-Based Data Dissemination (extended version). Technical Report TR: 2002-14, Univ. of Ioannina, Computer Science Dept, 2002.
15. R. Rastogi, S. Mehrotra, Y. Breitbart, H. F. Korth, and A. Silberschatz. On Correctness of Non-serializable Executions. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 97–108, 1993.
16. J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. In *ACM SIGMOD International Conference on Management of Data*, pages 85–96, 1999.
17. S. Shekar, D. Liu, "Genesis and Advanced Traveler Information Systems (ATIS): Killer Applications for Mobile Computing," MOBIDATA Workshop, New Jersey, 1994.
18. R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 236–246, 1985.
19. White Paper, Vitria Technology Inc. (http://www.vitria.com).
20. W. E. Weihl. Distributed Version Management for Read-Only Actions. *ACM Transactions on Software Engineering*, 13(1):56–64, 1987.
21. P. Xuan, S. Sen, O.J. Gonzalez-Gomez, J. Fernandez and K. Ramamritham, "Broadcast on Demand – Efficient and Timely Dissemination of Data in Mobile Environments," IEEE Real-Time Technology and Applications Symposium, pp. 38-48, June 1997.

# An Efficient Indexing Scheme for Multi-dimensional Moving Objects*

Khaled Elbassioni[1], Amr Elmasry[1], and Ibrahim Kamel[2]

[1] Computer Science Department, Alexandria University, Egypt
{elbassio,elmasry}@paul.rutgers.edu
[2] College of Information Systems, Zayed University, United Arab Emirates
Ibrahim.Kamel@zu.ac.ae

**Abstract.** We consider the problem of indexing a set of objects moving in $d$-dimensional space along linear trajectories. A simple disk-based indexing scheme is proposed to efficiently answer queries of the form: report all objects that will pass between two given points within a specified time interval. Our scheme is based on mapping the objects to a dual space, where queries about moving objects translate into polyhedral queries concerning their speeds and initial locations. We then present a simple method for answering such polyhedral queries, based on partitioning the space into disjoint regions and using a B-tree to index the points in each region. By appropriately selecting the boundaries of each region, we can guarantee an average search time that almost matches a known lower bound for the problem. Specifically, for a fixed $d$, if the coordinates of a given set of $N$ points are statistically independent, the proposed technique answers polyhedral queries, on the average, in $O((N/B)^{1-1/d}.(\log_B N)^{1/d} + K/B)$ I/O's using $O(N/B)$ space, where $B$ is the block size, and $K$ is the number of reported points. Our approach is novel in that, while it provides a theoretical upper bound on the average query time, it avoids the use of complicated data structures, making it an effective candidate for practical applications.

## 1 Introduction

Maintaining a database of moving objects arises in a wide range of applications, including air-traffic control, digital battlefields, and mobile communication systems [3,7]. Traditionally, a database management system assumes that data stored in the database remain constant until they are explicitly modified through an update. With moving objects, storing the continuously changing locations of these objects directly in the database becomes infeasible, considering the large update overhead. An obvious solution, to overcome this problem, is to represent each object by its parameters (velocity and initial location), which will be stored in the database, and update the database only when one of these

* The first and third authors gratefully acknowledge Panasonic Information and Networking Technologies Laboratory in Princeton, New Jersey, for its support during this work.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 425–439, 2003.

parameters changes. There has been some recent work on extending the current database technology to handle moving objects; see for example [18].

Given a database of $N$ objects moving in $d$-dimensional space along linear trajectories, we consider, in this paper, the problem of constructing an index on these objects to efficiently answer range queries over their locations in the future. An example of such queries, in a database of moving cars, is: "Report all cars that will pass through some given region, within the next ten minutes". It is assumed that each object moves along a straight line; an assumption that applies to a large class of problems such as cars in an almost straight-line highway. Furthermore, many non-linear functions can be approximated by connected straight line segments.

Specifically, assume that the position of an object moving with velocity vector $v = (v_1, \ldots, v_d)$ starting from location $a = (a_1, \ldots, a_d)$, at time $t \geq 0$, is given by $y = a + vt$. Given two locations $y', y'' \in \mathbb{R}^d$, and two time instances $t', t''$ (where $y' \leq y''$ and $t' \leq t''$), it is required to report all objects which will pass between these two locations, during the time period $[t', t'']$ (see Figure 1:a for the one-dimensional case). In other words, we are interested in reporting all moving objects whose coordinates in the $(d+1)$-dimensional space $(t, y_1, \ldots, y_d)$ satisfy

$$\begin{array}{ccccc} y'_i & \leq & y_i(t) = v_i t + a_i & \leq & y''_i, \text{ for } i = 1, \ldots, d, \\ t' & \leq & t & \leq & t'', \end{array} \tag{1}$$

where $y' = (y'_1, y'_2, \ldots, y'_d)$, $y'' = (y''_1, y''_2, \ldots, y''_d)$ and $y(t) = (y_1(t), \ldots, y_d(t))$.

In the standard external memory model of computation [4], the efficiency of an algorithm is measured in terms of the number of I/O's required to perform an operation. Let $B$ be the page size, i.e., the number of units of data that can be processed in a single I/O operation. If $K$ is the number of objects reported in a given query, then the minimum number of pages to store the data is $n \stackrel{\text{def}}{=} \lceil \frac{N}{B} \rceil$ and the minimum number of I/O's to report the answer is $k \stackrel{\text{def}}{=} \lceil \frac{K}{B} \rceil$. Thus the time and space complexity of a given algorithm, under such model, will be measured in terms of these parameters $n$ and $k$. Finally to simplify the presentation, we shall assume, when using the $O(\cdot)$ notation, that the dimension $d$ is constant.

The paper is organized as follows. In Section 2, we briefly survey related work. Section 3 states the main contribution of this paper and Section 4 gives an overview of the technique and the duality transformations used. We present the index structure and the search algorithm in Section 5, and the bound on the average query performance in Section 6. Preliminary experimental results on the one-dimensional case, comparing the performance of our method with other traditional techniques such as *R-trees*, are reported in Section 7. Finally, our conclusion is made in Section 8.

## 2 Related Work

One can generally distinguish two directions of research in the context of indexing moving objects. In the first one, techniques with theoretically guaranteed worst-

case query time have been developed. Most of these techniques resort to *duality* to transform queries about moving objects to polyhedral queries involving their parameters (velocities and initial locations). For the latter problem, Matoušek [15] gave an almost optimal main memory algorithm for simplex range searching using *partition trees*. Given a static set of $N$ points in $d$-dimensions and an $O(N)$ space, his technique answers any simplex range query in $O(N^{(d-1)/d+\epsilon} + K)$ time, after an $O(N \log N)$ preprocessing time, for any constant $\epsilon > 0$. For the lower bound, Chazelle and Rosenberg [8] showed that simplex reporting in $d$-dimensions, using only linear space, requires $\Omega(N^{(d-1)/d} + K)$ I/O's. These bounds have been adapted to the external memory model by Kollios et al. [12], implying in particular, a query time of $O(n^{1-1/2d+\epsilon} + k)$ I/O's for answering queries of type (1), using linear space. Building on partition trees, Agarwal et al. [1] were able to obtain an index for moving objects in two-dimensions, for which the worst-case query time is $O(n^{\frac{1}{2}+\epsilon}+k)$ I/O's using $O(n)$ space. However, these techniques might be practically inefficient since they use complicated data structures, and the constants hidden under their complexity bounds might be quite large if a small $\epsilon$ is sought. It should also be mentioned that if space is not an issue and is allowed to increase non-linearly with $n$, then logarithmic query time can be achieved; see [1,12] for examples. In the second approach, practical techniques have been developed that use the commercially available index structures such as B-trees, R-trees, R*-trees, etc [5,11,14]. Examples of this approach include the quad-tree method of [19], the time-parameterized R*-trees (*TPR trees*) of [16] and [6], and the work of Kollios et al. [12]. However, no theoretical bounds on the average query time were shown for these techniques.

In this paper, we propose a simple indexing structure that uses only B-trees in its implementation. We also provide an average case analysis of the query time given a random set of objects in the database. Our solution is based on a simple *internal memory* index structure, which we developed in [9], for answering polyhedral queries (i.e., queries determined by a finite set of linear constraints), whose average query time is as good as the worst-case query time obtained by the more complicated techniques. Such queries arise naturally in Spatial database applications; see [2,10] and the references therein.

## 3 The Contribution of the Paper

We can summarize the results of this paper as follows:

- Given a set of $N$ points in $\mathbb{R}^d$, we propose an index structure to enable efficient answering of polyhedral queries about these points. Under a natural assumption on the points coordinates, namely that they are statistically independent, we can answer a query, on the average, in $O(mn^{1-1/d}.(\log_B n)^{1/d} + mk)$ I/O's using $O(n)$ space, where $m$ is the number of linear constraints bounding the query region. Moreover, this result is valid for any data distribution (not necessarily uniform), and does not require the distribution function to be explicitly given. This gives an upper bound that almost matches

the worst-case bound obtained by the more complicated algorithms such as [15]. However, our algorithm is much simpler since it requires only B-trees for its implementation. Moreover, the query algorithm works directly on the original query region, and does not require partitioning it into simplices as is usually required by other algorithms.
- For a set of $N$ objects moving in one-dimensional space, with any distribution of velocities and initial locations, we use the above method in the dual space to answer queries of type (1), on the average, in $O(\sqrt{n \log_B n} + k)$ I/O's using $O(n)$ space.
- For a set of $N$ objects moving in $d$-dimensional space, with uniform distributions of velocities and initial locations, we answer queries (1), on the average, in $O(n^{1-1/3d}.(\log_B N)^{1/3d} + k)$ I/O's using $O(n)$ space.

## 4 Duality Transformations

One of the main challenges in indexing moving objects is that the trajectories of the objects are monotonically increasing with time. Thus, representing each object by a Minimum Bounding Box (MBB) is not appropriate since the overlap between the MBB's can be excessive, leading to bad performance. It is more efficient to represent each object by its parameters $v_i$ and $a_i$ and transform queries about moving objects into queries concerning these parameters. One commonly used transformation is to map each object with velocity vector $v = (v_1, \ldots, v_d)$ and initial location $a = (a_1, \ldots, a_d)$ into a $2d$-dimensional point $(v_1, a_1, \ldots, v_d, a_d)$ (for $d = 1$, this is called the Hough-X transform in [13]). Under this transformation, query region (1) becomes a $2d$-dimensional region, bounded by a set of linear constraints for $d = 1$, and a mix of linear and quadratic constraints for $d \geq 2$ (see Figure 1:b):

**Proposition 1.** *The d-dimensional query (1) can be expressed in the* $(v_1, a_1, \ldots, v_d, a_d)$ *space as follows:*

**(i)** *The d constraints* $\begin{cases} y_i' - v_i t'' \leq a_i \leq y_i'' - v_i t', & \text{if } v_i > 0 \\ y_i' - v_i t' \leq a_i \leq y_i'' - v_i t'', & \text{if } v_i \leq 0 \end{cases}$ *for* $i = 1, \ldots, d$.

**(ii)** *The set of* $\binom{d}{2}$ *constraints, for* $1 \leq i < j \leq d$*:*

$$\begin{cases} y_i' v_j - y_j'' v_i \leq a_i v_j - a_j v_i \leq y_i'' v_j - y_j' v_i, & \text{if } v_i > 0 \text{ and } v_j > 0 \\ y_i'' v_j - y_j'' v_i \leq a_i v_j - a_j v_i \leq y_i' v_j - y_j' v_i, & \text{if } v_i > 0 \text{ and } v_j \leq 0 \\ y_i' v_j - y_j' v_i \leq a_i v_j - a_j v_i \leq y_i'' v_j - y_j'' v_i, & \text{if } v_i \leq 0 \text{ and } v_j > 0 \\ y_i'' v_j - y_j' v_i \leq a_i v_j - a_j v_i \leq y_i' v_j - y_j'' v_i, & \text{if } v_i \leq 0 \text{ and } v_j \leq 0. \end{cases}$$

*Proof.* Use the Fourier-Motzkin elimination method [17] to eliminate the variable $t$ from the set of constraints in (1). □

Since our proposed index can only deal with linear constraints, we shall get rid of such quadratic constraints by resorting to another type of duality transformation. Specifically, assuming $v_i \neq 0$ (for objects having $v_i = 0$, the problem can be reduced to a lower-dimensional problem; see Section 6), we let

$u_i \stackrel{\text{def}}{=} 1/v_i$ and $w_i \stackrel{\text{def}}{=} a_i/v_i$ be the new transformed parameters (this is the so-called Hough-Y transform in [13]). Note that for $v_i > 0$, this gives a one-to-one correspondence between $(v_i, a_i)$ and $(u_i, w_i)$. Thus, with such a transformation, our query region becomes a $2d$-dimensional polyhedron:

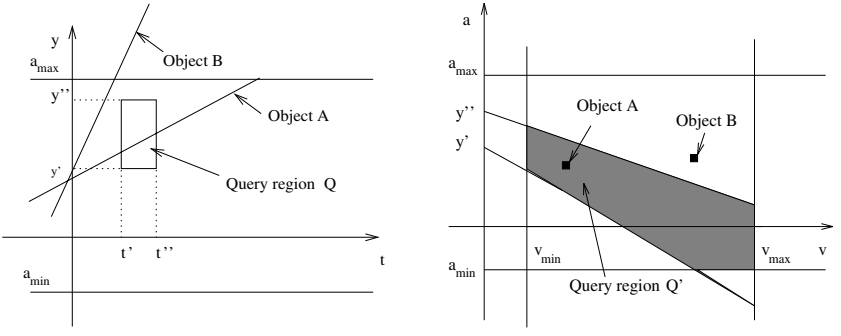
**Proposition 2.** *The d-dimensional query (1) can be expressed in the* $(u_1, w_1, \ldots, u_d, w_d)$*-space as follows:*

**(i)** *The d constraints* $\begin{cases} y_i' u_i - t'' \le w_i \le y_i'' u_i - t', & \text{if } u_i > 0 \\ y_i'' u_i - t'' \le w_i \le y_i' u_i - t', & \text{if } u_i \le 0 \end{cases}$ *for* $i = 1, \ldots, d$.

**(ii)** *The set of* $\binom{d}{2}$ *constraints, for* $1 \le i < j \le d$:

$$\begin{cases} y_i' u_i - y_j'' u_j \le w_i - w_j \le y_i'' u_i - y_j' u_j, & \text{if } u_i > 0 \text{ and } u_j > 0 \\ y_i' u_i - y_j' u_j \le w_i - w_j \le y_i'' u_i - y_j'' u_j, & \text{if } u_i > 0 \text{ and } u_j < 0 \\ y_i'' u_i - y_j'' u_j \le w_i - w_j \le y_i' u_i - y_j' u_j, & \text{if } u_i < 0 \text{ and } u_j > 0 \\ y_i'' u_i - y_j' u_j \le w_i - w_j \le y_i' u_i - y_j'' u_j, & \text{if } u_i < 0 \text{ and } u_j < 0 \ . \end{cases}$$

However, there is a problem with such transformation: the bound on the performance of our index uses the independence assumption. While it is natural to assume that the parameters $a_1, v_1, \ldots, a_d, v_d$ are statistically independent, this is not the case for the transformed variables $u_1, w_1, \ldots, u_d, w_d$. We handle this problem in Section 6: we use a property of the Hough-Y transform to show how the index structure can be modified in this case.




a: Native space. b: Dual space.

**Fig. 1.** Query region in the native and dual spaces for $d = 1$.

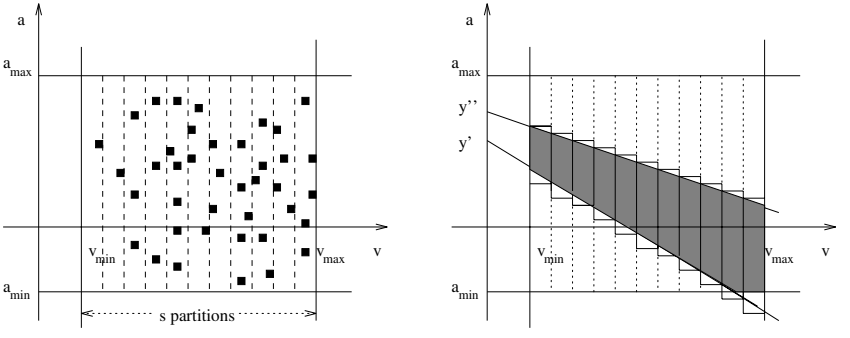
Having transformed the query region into a polyhedron, we develop, in the next section, an index structure, which we call *the MB-index*, to answer such queries efficiently. Thus, to summarize, for $d = 1$, we use the Hough-X duality transform and the MB-index in the 2-dimensional space $(v_1, a_1)$. For $d \ge 2$, we use the Hough-Y transform and the MB-index in the $2d$-dimensional space $(u_1, w_1, \ldots, u_d, w_d)$.

## 5 MB-Index for Answering Polyhedral Queries in $\mathbb{R}^d$

Given a set $\mathcal{P}$ of $N$ $d$-dimensional points distributed in a rectangular box $\mathcal{B} = [L_1, U_1] \times \ldots \times [L_d, U_d] \subseteq \mathbb{R}^d$. The location $x = (x_1, \ldots, x_d)$ of each point can be regarded as a random variable distributed in the interval $L_1 \leq x_1 \leq U_1$, $\ldots, L_d \leq x_d \leq U_d$, with probability density function $f(x_1, \ldots, x_d)$. It is natural to assume that these locations are statistically independent along each direction, i.e., $f(x_1, \ldots, x_d) = \prod_{i=1}^{d} f_i(x_i)$, where $f_i(x_i)$ is the probability density function along the $i$th direction. We further assume without loss of generality that the points are in general position.

Given a set of half-spaces determined by the hyperplanes $\mathbb{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_m\}$, it is required to report all the points that lie in the intersection of these half-spaces. In this section, we describe an index structure which enables us to efficiently answer such polyhedral queries about the given point set.

**Index structure and search algorithm.** The general idea is to partition the space into disjoint rectangular regions, and index the points of each region on one of the dimensions using a B-tree. Given a query polyhedron, the polyhedron will be intersected with each region, and the intersection will be approximated by a Minimum Bounding Box (MBB) (see Figure 2). Finally, the points in each MBB are reported including possibly some false hits. By selecting the region boundaries so as to balance the number of points in each region, we can guarantee that the number of false hits is not too large.




a: Partitioning the space. b: Partitioning the query region.

**Fig. 2.** Partitioning the space and the query region.

In more details, to build the MB-index, we fix one dimension, say the $d$th dimension, and split the box $\mathcal{B}$ in each other dimension $i = 1, \ldots, d-1$, using $s$ $(d-1)$-dimensional hyperplanes, perpendicular to the $i$th dimension. To minimize the number of false hits (see Lemma 1 below), we select the number of partitions $s$ as follows:

$$s = \left( \frac{n}{\log_B n} \right)^{\frac{1}{d}}.$$

Let $\mu_i(j)$ be the point on the $i$th axis defining $j$th hyperplane in the $i$th dimension (i.e., the hyperplane is $\{x \in \mathbb{R}^d \mid x_i = \mu_i(j)\}$), where $i = 1, \ldots, d-1$ and $j \in [s] \stackrel{\text{def}}{=} \{0, 1, \ldots, s-1\}$. Thus $\mu_i(0) = L_i$ and $\mu_i(s) = U_i$ for $i = 1, \ldots, d-1$. To bound the error probability, these hyperplanes will be chosen such that for all $i$ and $j$

$$\int_{\mu_i(j)}^{\mu_i(j+1)} f_i(x_i)dx_i = \frac{1}{s} \tag{2}$$

is satisfied. Clearly, if the distributions of the coordinates are uniform, then the resulting partitions will be equi-distant. In general, numerical (or analytical) integration can be used to obtain the required partitioning as described by (2), for any density function $f_i(x_i)$. Practically speaking, (2) says that the number of points in each interval should be $1/s$ of the total number $N$. Thus, for $i = 1, \ldots, d-1$ and $j \in [s]$, the region boundary $\mu_i(j)$ can be selected so as to make $|\{p \in \mathcal{P} \mid \mu_i(j) \leq p_i \leq \mu_i(j+1)\}| = N/s$.

Thus, with the above partitioning, we obtain a set of $s^{d-1}$ sub-boxes $\{\mathcal{B}_J \mid J \in \mathcal{J}\}$, where $\mathcal{J} \stackrel{\text{def}}{=} \{(j_1, \ldots, j_{d-1}) \mid j_i \in [s], \text{ for all } i = 1, \ldots, d-1\}$, and $\mathcal{B}_J = \mathcal{B}_{j_1,\ldots,j_{d-1}}$ is the sub-box of $\mathcal{B}$ defined by the two corner points $(\mu_1(j_1), \ldots, \mu_{d-1}(j_{d-1}))$ and $(\mu_1(j_1+1), \ldots, \mu_{d-1}(j_{d-1}+1))$. A B-tree $\mathcal{T}(\mathcal{B}_J)$ is then constructed to index the points in each such sub-box $\mathcal{B}_J$. Obviously, the points in each of these B-trees are ordered by the value of their $d$th coordinate.

For a hyperplane $\mathcal{H} = \{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d = c\}$, let us denote respectively by $\mathcal{H}^-$ and $\mathcal{H}^+$ the closed half-spaces $\{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d \leq c\}$ and $\{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d \geq c\}$. Suppose we are interested in reporting all the points that lie on one side of $\mathcal{H}$, say $\mathcal{H}^-$. To do this, we intersect $\mathcal{H}$ with each of the sub-boxes $\mathcal{B}_J$. The highest point, with respect to the $d$th coordinate, in each intersection is determined, and all the points that lie below this point (have smaller value in the $d$th dimension) are reported. Next, each of these points is checked, and only accepted if it lies in the required half-space defined by $\mathcal{H}$. More precisely, for $i = 1, \ldots, d-1$, define

$$\alpha_i = \begin{cases} 1 & \text{if } a_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Then the $d$th coordinates of the lowest and highest point in $\mathcal{B}_J$ that intersect $\mathcal{H}$, for $J = (j_1, \ldots, j_{d-1}) \in \mathcal{J}$, are given by

$$\lambda(J) = \frac{c}{a_d} - \sum_{i=1}^{d-1} \frac{a_i}{a_d} \cdot \mu_i(j_i + \alpha_i), \qquad \Lambda(J) = \frac{c}{a_d} - \sum_{i=1}^{d-1} \frac{a_i}{a_d} \cdot \mu_i(j_i + 1 - \alpha_i), \tag{4}$$

provided $a_d \neq 0$. If $a_d = 0$, then $\lambda(J), \Lambda(J)$ are set to $\pm\infty$, depending on the signs of $c - \sum_{i=1}^{d-1} a_i \cdot \mu_i(j_i + \alpha_i)$, $c - \sum_{i=1}^{d-1} a_i \cdot \mu_i(j_i + 1 - \alpha_i)$, respectively.

Now, given a set $\mathbb{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_m\}$ of hyperplanes, where $\mathcal{H}_q = \{x \in \mathbb{R}^d \mid a_1^q x_1 + \ldots + a_d^q x_d = c^q\}$, let us assume, without loss of generality that,

$a_d^q \geq 0$ for $r = 1, \ldots, m$. Suppose it is required to report all the points that lie in the intersection $\left(\bigcap_{q\in Q^+} \mathcal{H}_q^+\right) \bigcap \left(\bigcap_{q\in Q^-} \mathcal{H}_q^-\right)$, where $Q^+ \cup Q^- = \{1, \ldots, m\}$. In the following procedure, we denote by $\alpha_i^q, \lambda^q, \Lambda^q$, the values of these parameters as computed, using (3), (4), with respect to the hyperplane $\mathcal{H}_q$.

**Search Algorithm:**
Input: An MB-index containing a set $\mathcal{P} \subseteq \mathbb{R}^d$ of $n$ points, a set of $m$ hyperplanes $\mathbb{H}$, and a partition $Q^+ \cup Q^-$ of $\{1, \ldots, m\}$.
Output: All points that lie in $\left(\bigcap_{q\in Q^+} \mathcal{H}_q^+\right) \bigcap \left(\bigcap_{q\in Q^-} \mathcal{H}_q^-\right)$.

For each $J \in \mathcal{J}$
1. Compute $\lambda^q(J), \Lambda^q(J)$, for $q = 1, \ldots, m$ as described above.
2. Let $l \leftarrow \text{argmax}\{\lambda^q(J) \mid q \in Q^+\}$ [a], $r \leftarrow \text{argmin}\{\Lambda^q(J) \mid q \in Q^-\}$.
3. Search the B-tree $\mathcal{T}(\mathcal{B}_J)$ for the set of candidates
$$\mathcal{C} = \{p \in \mathcal{T}(\mathcal{B}_J) \mid \lambda^l(J) \leq p_d \leq \Lambda^r(J)\}$$
inside the sub-box $\mathcal{B}_J$ satisfying the query.
4. For each $p \in \mathcal{C}$, if $p$ lies inside the query polyhedron, report $p$.

[a] where argmin means an index at which the minimum value is attained

It is easy to see that the above procedure is conservative in the sense that no false dismissals are possible. On the other hand, using the assumption about statistical independence and our partitioning strategy (2), the following key lemma, which is proved in [9], follows:

**Lemma 1.** *For any half-space query, the expected number of false hits is at most* $(d-1)\frac{n}{s}$ *I/O's.*

**Insertions/Deletions.** Finally, let us explain how to make our index dynamic. Clearly, each insertion/deletion requires $O(\log_B n)$ I/O's. However, after a number of updates, say insertions, the number of points in some trees may increase significantly, possibly degrading the query performance. In that case, every overcrowded tree is split into smaller trees. Since the split cost is non-trivial, it is reasonable to split only when the number of objects in the tree increases by some predefined factor. Similarly, if after deleting an object from a given tree, the number of objects drops below some factor of the original number, then the tree is merged with one or more adjacent trees. Moreover, when the total number of points increases/decreases by some predefined factor of the original number, the whole structure is rebuilt. This way the insert/delete operation will only take $O(\log_B n)$ I/O's in the amortized sense.

Using Lemma 1, we get the following result about the performance of the MB-index.

**Theorem 1.** *Under the statistical independence assumption, the average number of I/O's required by the MB-index to report all the points in the intersection of* $m$ *half-spaces is* $O(mn^{1-\frac{1}{d}} \log_B^{\frac{1}{d}} n + mk)$. *The space required is* $O(n)$, *the preprocessing is* $O(N \log_B n)$, *and the amortized update time is* $O(\log_B n)$.

## 6 Bounding the Average Query Performance

As stated earlier, the bound on the MB-index performance is based on the independence assumption. However, this assumption can be violated if we use the transformation $u_i = \frac{1}{v_i}$, $w_i = \frac{a_i}{v_i}$. In this section, we discuss how to handle this problem.

Let us consider moving objects whose speeds and intercepts $v = v_i$ and $a = a_i$ in a given direction $i$ are independent and uniformly distributed in the intervals $[v_{min}, v_{max}]$ and $[a_{min}, a_{max}]$ respectively. Assume without loss of generality that $a_{min} = -a_{max}$ (by translating the points along the $a$ axis), and that $v_{min} > 0$ (points with $v_{min} < 0$ are handled similarly, while points with $v \approx 0$ are considered fixed and handled separately).

To be able to use the results of the previous section, we show in the Appendix that, the Hough-Y transform exhibits a nice property. Namely, for independent and uniformly distributed $a, v$, and for any $\eta' \le \eta'', \mu' \le \mu''$, the variables $u = 1/v$ and $w = a/v$ satisfy the inequalities

$$\int_{u=\eta'}^{\eta''} \int_{w=\mu'}^{\mu''} f(u,w)dudw \le \begin{cases} 4 \int_{\eta'}^{\eta''} g(u)du \int_{\mu'}^{\mu''} h(w)dw, & \text{if } \mu' \le \frac{a_{max}}{\mathbb{E}[v]} \text{ and } \mu'' \ge \frac{a_{min}}{\mathbb{E}[v]} \\ \frac{1}{4}\left(1 - \frac{v_{min}}{v_{max}}\right) \int_{\eta'}^{\eta''} g(u)du, & \text{otherwise,} \end{cases} \quad (5)$$

where $g$ and $h$ are respectively the probability density functions of $u$ and $w$, $f$ their joint density function, and $\mathbb{E}[v] = (v_{max} + v_{min})/2$ is the average speed.

Then, using the MB-index technique of the previous section, we fix one direction, say $u_d$, and partition the space along each direction $u_i$, $i = 1, \ldots, d-1$ into $s$ intervals determined by the points $\eta_i(0), \ldots, \eta_i(s)$, such that $\int_{\eta_i(j)}^{\eta_i(j+1)} g_i(u_i)du_i = 1/s$, where $s = (n/\log_B n)^{1/(2d)}$. Similarly, we partition the space along each direction $w_i$, $i = 1, \ldots, d$ into $s$ intervals determined by the points $\mu_i(0), \ldots, \mu_i(s)$, such that $\int_{\mu_i(j)}^{\mu_i(j+1)} h_i(w_i)dw_i = 1/s$. If for every $i = 1, \ldots, d$ we further have $v_{i,min}/v_{i,max} \ge 1 - 1/s$, then (5), applied to $u_i, w_i$ for $i = 1, \ldots, d$, would imply that the probability of a false hit in any sub-region is upper-bounded as follows:

$$\begin{aligned} \Pr[FH] &= \int_{\eta_1(j_1)}^{\eta_1(j_1+1)} \int_{\mu_1(j_1')}^{\mu_1(j_1'+1)} \cdots \int_{\eta_d(j_d)}^{\eta_d(j_d+1)} \int_{\mu_d(j_d')}^{\mu_d(j_d'+1)} f(u_1, w_1, \ldots, u_d, w_d)du_1dw_1 \ldots du_ddw_d \\ &\le 4^d \prod_{i=1}^{d} \int_{\eta_i(j_i)}^{\eta_i(j_i+1)} g_i(u_i)du_i \prod_{i=1}^{d} \int_{\mu_i(j_i')}^{\mu_i(j_i'+1)} h_i(w_i)dw_i. \end{aligned}$$

Consequently, we conclude by summing up all these probabilities that the total probability of error is upper-bounded by $\frac{4^d}{s}$ (see the proof of Lemma 1 in [9]). However, for $i = 1, \ldots, d$, the interval $[v_{i,min}, v_{i,max}]$ may be large making the assumption $v_{i,min}/v_{i,max} \ge 1 - 1/s$ invalid. To solve this problem, we partition this interval into $k = \lceil s \ln(v_{i,max}/v_{i,min}) \rceil$ intervals $[v_i(0), v_i(1)], \ldots, [v_i(k-1), v_i(k)]$, where $v_i(0) = v_{i,min}$, $v_i(k) = v_{i,max}$, and $v_i(j+1) = v_i(j).s/(s-1)$ for $j = 0, \ldots, k-1$. Clearly, in each such interval, the variable $v_i$ is still uniformly distributed and $v_i(j)/v_i(j+1) \ge 1 - 1/s$, as required. Performing such a a partitioning for $i = 1, \ldots, d$, we obtain at most $\sigma s^d$ different regions, where $\sigma \stackrel{\text{def}}{=} \prod_{i=1}^{d} \ln(v_{i,max}/v_{i,min})$.

Thus to summarize, given a set of $N$ $d$-dimensional moving objects, the index construction goes as follows:
1. Partition the set of points into $3^d$ groups according to whether $v_i < 0$, $v_i = 0$, or $v_i > 0$, for $i = 1, \ldots, d$.
2. Within each group, further partition the points, as explained above, into $\sigma s^d$ groups, according to which region of $[v_i(0), v_i(1)], [v_i(1), v_i(2)], \ldots$ contains $v_i$ for $i = 1, \ldots, d$.
3. Finally, for each of the resulting groups, represent the points in the $(u_1, w_1, \ldots, u_d, w_d)$ space, and use an MB-index (consisting of $s^{2d-1}$ B-trees) to store them.

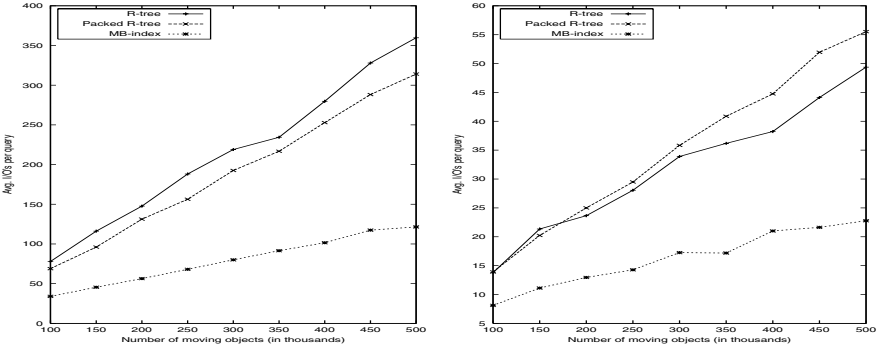
Let us now estimate the average query time of the resulting index. The total number of B-trees used is $\sigma(3s)^d.s^{2d-1} = \sigma 3^d s^{3d-1}$. The probability of error in each tree is at most $4^d/s$ as pointed out above. Thus selecting $s = \left(\frac{n}{\sigma \log_B n}\right)^{\frac{1}{3d}}$, we achieve an average search time of $O(n^{1-\frac{1}{3d}}(\sigma \log_B n)^{\frac{1}{3d}} + k)$, for a fixed $d$. Of course, for $d = 1$, we can use the MB-index directly in the $(v_1, a_1)$-space, and achieve an average query time of $O(\sqrt{n \log_B n} + k)$. We summarize our results in the following Theorem.

**Theorem 2.** *(i) For a set of $N$ objects moving in one-dimensional space, with statistically independent velocities and initial locations, we can use the MB-index method to answer queries (1), on the average, in $O(\sqrt{n \log_B n} + k)$ I/O's using $O(n)$ space. The preprocessing is $O(N \log_B n)$, and the amortized update time is $O(\log_B n)$.*
*(ii) For a set of $N$ objects moving in $d$-dimensional space, with uniformly distributed and independent velocities and initial locations, queries (1) can be answered, on the average, in $O(n^{1-1/3d}(\sigma \log_B n)^{1/3d} + k)$ I/O's using $O(n)$ space. The preprocessing is $O(N \log_B n)$, and the amortized update time is $O(\log_B n)$.*

## 7 Experimental Results

In this section, preliminary experimental results on the proposed indexing approach for the one-dimensional case are presented, and compared against R-tree based methods. As stated earlier, representing each object by a minimum bounding box (MBB) and using MBB-based indexing structures, such as R-trees, is not appropriate since the overlap between the MBB's will be excessive. Thus, to have a fair comparison with R-tree techniques, we first mapped each object into the dual space, where the query region becomes a trapezoid. Then we used the algorithm proposed in [10] to perform the intersection test between a trapezoid and a rectangle. For comparison purposes, two variants of R-trees have been used. The first is Guttman's R-tree [11] with quadratic splitting strategy. Objects were inserted incrementally, one at a time, so no preprocessing of the data was done. In the second variant, we used a *Packed R-tree*, where data is packed in the leaves of the tree in such a way to improve both node utilization and query processing time [14].

In the experiments, we used both uniform and non-uniform distribution of objects. In each case, two query sizes were used: 8% and 1% of the total space. The average number of I/O's over 1000 uniformly generated queries were then computed for each indexing technique. The page size used is 4096 bytes.




a: Query size=8% of space. b: Query size=1% of space.

**Fig. 3.** Average query performance for various indexing techniques with Uniform distribution.

In the first set of experiments, we generated $N = 100K, 200K, \ldots, 500K$ uniformly distributed objects, where each object was generated by picking, at random, its velocity $v$ and its starting location $a$, where $v \in [0.16, 1.66]$, $a \in [0, 200]$. All speeds were assumed positive for simplicity. Figures 3:a,b present our results for the two types of queries. In these two figures, we show the average number of I/O's per query versus the number of objects in the database. As can be seen from the figures, the MB-index approach outperformed the R-tree based techniques. This gain in performance (approximately a factor of 2.5) is contributed to by two main components. The first is that the MB-index minimizes the dead space, so the number of I/O's can be significantly smaller. The second component is the fan-out of tree nodes. For the same page size, the B-tree has a larger fan-out than the R-tree since the B-tree is built on one dimensional data, while each entry in the R-tree contains a rectangle.

In the second experiment, we examined the effect of changing the distribution of speeds on the average number of I/O's for various techniques. For this experiment a normal distribution of speeds with mean $(v_{min} + v_{max})/2$ and standard deviation 1 was used. Table 1 presents the results for 8% and 1% queries. In the table, we show the average number of I/O's for different values of $N$. As seen from the table, while the performance of the simple R-tree degrades significantly with such distribution, both the packed R-tree and the MB-index methods remain almost invariant. The MB-index method remains superior to the packed R-tree technique for this case too.

**Table 1.** Average query performance for Normal distribution.

| $N$ | I/O's for query size=8% | | | I/O's for query size=1% | | |
|---|---|---|---|---|---|---|
| | MB-index | R-tree | Packed R-tree | MB-index | R-tree | Packed R-tree |
| 100K | 31.908 | 489.998 | 68.281 | 8.553 | 244.096 | 14.634 |
| 150K | 43.964 | 729.965 | 98.476 | 13.016 | 372.229 | 18.845 |
| 200K | 56.232 | 966.029 | 128.455 | 13.870 | 493.500 | 24.282 |
| 250K | 68.163 | 1209.458 | 157.413 | 14.926 | 624.128 | 29.860 |
| 300K | 83.375 | 1441.948 | 190.943 | 18.961 | 740.153 | 33.986 |
| 350K | 92.011 | 1688.847 | 217.671 | 21.987 | 868.695 | 39.315 |
| 400K | 100.383 | 1933.160 | 252.462 | 22.858 | 990.933 | 45.241 |
| 450K | 118.355 | 2146.008 | 280.440 | 22.529 | 1110.461 | 49.073 |
| 500K | 140.457 | 2394.094 | 306.693 | 26.791 | 1245.052 | 54.106 |

## 8 Conclusion

In this paper we proposed a new technique for indexing objects moving in $d$-dimensional space along straight lines. We showed that this technique exhibits an efficient average query time under moderate assumptions on the object distributions. A by-product of our technique is an efficient method for indexing multi-dimensional queries determined by linear constraints. One distinguishing feature of our index is its simplicity which makes it practically applicable. Experimental results indicate that this technique outperforms, by a factor of almost 2.5, the performance of other traditional methods based on R-trees. In future work, we intend to address extensions for other types of queries.

## References

1. P. K. Agarwal, L. Arge and J. Erickson, Indexing Moving points. In *Proc. 19th ACM PODS*, pp. 175–186, 2000.
2. P. K. Agarwal, L. Arge, J. Erickson, P. Franciosa and J. S. Vitter, Efficient Searching with Linear Constraints. In *Proc. 17th ACM PODS* , pp. 169–178, 1998.
3. R. Alonso and H. F. Korth, Database System Issues in Nomadic Computing. In *Proc. ACM-SIGMOD International Conference on Management of Data* , pp. 388–392, 1993.
4. A. Aggarwal and J. S. Vitter, The Input/Output Complexity of Sorting and Related Problems. In *Communications of the ACM*, 31(9):1116–1127, 1988.
5. N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, The $R^*$-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM-SIGMOD*, pp. 322–331, 1990.
6. M. Cai, D. Keshwani and P. Z. Revesz, Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology*, LNCS 1777, pp. 430–444. Springer, 2000.
7. S. Chamberlain, Model-based battle command: A paradigm whose time has come. In *Proc. 1st International Symposium on Command and Control Research and Technology*, pp. 31–38, 1995.

8. B. Chazelle and B. Rosenberg, Lower Bounds on the Complexity of Simplex Range Reporting on a Pointer Machine. In *Proc. 19th International Colloquium on Automata, Languages and Programming*, LNCS, Vol. 693, 1992.
9. K. Elbassioni, A. Elmasry and I. Kamel, Efficient Answering of Polyhedral Queries in $\mathbb{R}^d$ using BBS-trees. In *Proc. 14th Canadian Conference on Computational Geometry (CCCG 2002)*, pp. 54–57, 2002.
10. J. Goldstein, R. Ramakrishnan, U. Shaft and J. B. Yu, Processing Queries by Linear Constraints. In *Proc. 16th ACM PODS*, pp. 257–267, 1997.
11. A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM-SIGMOD*, pp. 47–57, 1984.
12. G. Kollios, D. Gunopulos and V. Tsotras, On Indexing Mobile Objects. In *Proc. 18th ACM PODS,* pp. 261–272, 1999.
13. H. V. Jagadish, On Indexing Line Segments. In *Proc. 16th VLDB Conference* , pp. 614–625, 1990.
14. I. Kamel and C. Faloutsos, On Packing R-trees. In *Proc. Second International Conference on Information and Knowledge Management*, 1993.
15. J. Matoušek, Efficient Partition Trees. *Disc. and Computational Geometry*, 8 (1992), pp. 432–448.
16. S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, Indexing the Positions of Continuously Moving Objects. In *Proc. ACM-SIGMOD*, pp. 331–342, 2000.
17. A. Schrijver. *Theory of Linear and Integer Programming*, Wiley-Interscience, 1986.
18. A. P. Sistla, O. Wolfson, S. Chamberlain and S. Dao, Modeling and Querying Moving Objects. In *Proc. 13th IEEE ICDE Conference*, pp. 422–432, April 1997.
19. J. Tayeb, O. Ulusoy and O. Wolfson, A quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

## Appendix: Proof of Inequality (5)

Define $L = \int_{u=\eta'}^{\eta''} \int_{w=\mu'}^{\mu''} f(u,w)dudw$ and $R = \int_{u=\eta'}^{\eta''} g(u)du \int_{w=\mu'}^{\mu''} h(w)dw$, and note that

$$L = \Pr[\frac{1}{\eta''} \le v \le \frac{1}{\eta'}, \mu' v \le a \le \mu'' v],$$
$$R = \Pr[\frac{1}{\eta''} \le v \le \frac{1}{\eta'}]. \Pr[\mu' v \le a \le \mu'' v].$$

By uniformity and independence of $v, a$, these probabilities are proportional to the areas of the corresponding regions in the $(v, a)$ space (in particular, $L$ is equal to the area of the shaded region in Figure 4). We consider 7 cases according to how the lines $a = \mu' v$ and $a = \mu'' v$ cross the borders of the probability region (Figures 4:a,b, and 8:c-g, respectively). Any other case is implied by, or can be reduced to, one of these cases. Below, we use for brevity the notation $h_t = v_{max} - v_{min}$ and $a_t = a_{max} - a_{min}$. The symbols $x, x', y, y', y'', h', h''$, and $h'''$, in each case, denote the distances shown in the corresponding figure.

*Case 1.* $L = \frac{(x+y)h}{2a_t h_t}$, $R = \frac{(x'+y')h_t}{2a_t h_t} \cdot \frac{h}{h_t}$. Then $\frac{L}{R} = \frac{x+y}{x'+y'} < 2$. (see Figure 4:a.)

*Case 2.* $\frac{L}{R} = \frac{2x'}{2x'h'/h_t+(x'+y)(h''-h')/h_t+(y+y')(1-h''/h_t)}$. Since $\frac{y}{x'} \geq 1 - \frac{h''}{h_t}$, we get $\frac{L}{R} \leq \frac{2x'}{x'(h'+h'')/h_t+x'(1-h'/h_t)(1-h''/h_t)+y'(1-h''/h_t)} < 2$.

*Case 3.* $\frac{L}{R} = \frac{2x'h'/h+(x'+x)(h''-h')/h+(x+y)(h-h'')/h}{2x'h'/h_t+(x'+x)(h''-h')/h_t+(x+y')(1-h''/h_t)} < 2$, since $\frac{x}{x'} \geq 1 - \frac{h''}{h_t}$.

For cases 4,5, and 6 below, assume first that $\mu'' \geq \frac{a_{min}}{\mathbb{E}[v]}$.

*Case 4.* $\frac{L}{R} = \frac{x'+x}{yh'/h_t+y'(h'-h'')/h_t}$. Using the assumption $\mu'' \geq \frac{a_{min}}{\mathbb{E}[v]}$, we get $h'/h_t \geq 1/2$, implying that $\frac{L}{R} \leq \frac{2y}{y/2+y'(h'-h'')/h_t} < 4$.

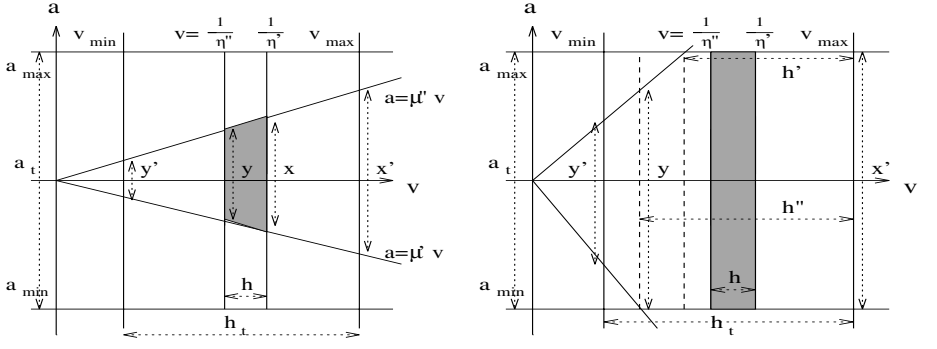
*Case 5.* $\frac{L}{R} = \frac{(x+y)}{x'h''/h_t+(x'+y')(h'-h'')/h_t} \leq \frac{2x'}{x'h'/h_t+y'(h'-h'')/h_t} < 4$.

*Case 6.* $\frac{L}{R} = \frac{(x'+x)(h''-h''')/h+(x+y)(h'''+h-h'')/h}{xh''/h_t+(x+y')(h'-h'')/h_t} \leq \frac{2x}{xh'/h_t+y'(h'-h'')/h_t} < 4$.

On the other hand, if $\mu'' < \frac{a_{min}}{\mathbb{E}[v]}$, that is, if $h'/h_t < 1/2$, then as we can see in Figure 4:g, the probability $L$ is bounded by $h/h'$ times the ratio of the area of the shaded region to the total area of the probability space. Noting that $a_{min} = -a_{max}$ and $h/h_t = \int_{u=\eta'}^{\eta''} g(u)du$, we conclude that

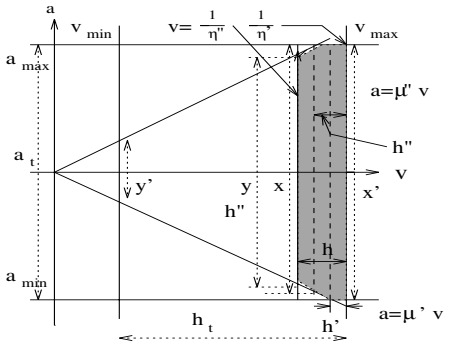
$$L \leq \frac{(\mu'' v_{min} - a_{min})h}{2a_t h_t} < \frac{1}{4}\left(1 - \frac{v_{min}}{\mathbb{E}[v]}\right).\frac{h}{h_t} < \frac{1}{4}\left(1 - \frac{v_{min}}{v_{max}}\right)\int_{u=\eta'}^{\eta''} g(u)du$$

as desired. □

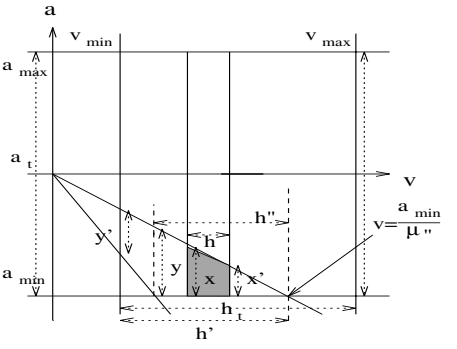



a: Case 1. b: Case 2.

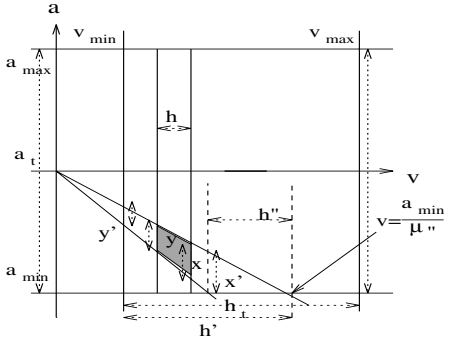
**Fig. 4.** The case analysis proof of Inequality (5).



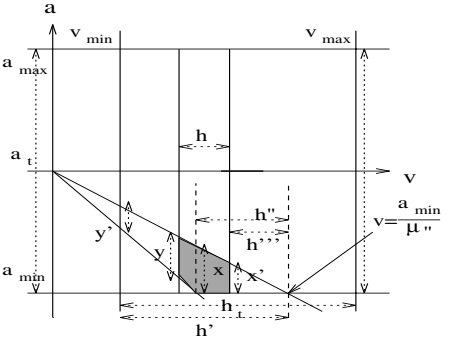

c: Case 3.



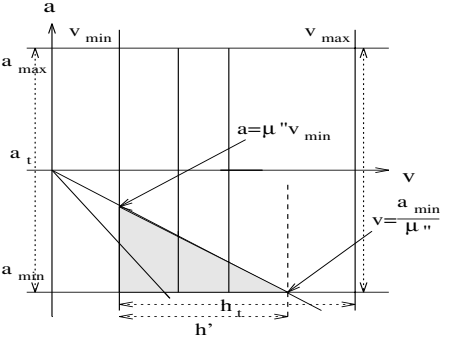

d: Case 4.




e: Case 5.




f: Case 6.




g: Case $\mu'' < \frac{a_{min}}{\mathbb{E}[v]}$.

**Fig. 4** (Cont.) The case analysis proof of Inequality (5).

# Nearest Neighbors Can Be Found Efficiently If the Dimension Is Small Relative to the Input Size

Michiel Hagedoorn

Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany,
hagedoorn@mpi-sb.mpg.de

**Abstract.** We consider the problem of nearest-neighbor search for a set of $n$ data points in $d$-dimensional Euclidean space. We propose a simple, practical data structure, which is basically a directed acyclic graph in which each node has at most two outgoing arcs. We analyze the performance of this data structure for the setting in which the $n$ data points are chosen independently from a $d$-dimensional ball under the uniform distribution. In the average case, for fixed dimension $d$, we achieve a query time of $O(\log^2 n)$ using only $O(n)$ storage space. For variable dimension, both the query time and the storage space are multiplied with a dimension-dependent factor that is at most exponential in $d$. This is an improvement over previously known time–space tradeoffs, which all have a super-exponential factor of at least $d^{\Theta(d)}$ either in the query time or in the storage space. Our data structure can be stored efficiently in secondary memory: In a standard secondary-memory model, for fixed dimension $d$, we achieve average-case bounds of $O((\log^2 n)/B + \log n)$ query time and $O(N)$ storage space, where $B$ is the block-size parameter and $N = n/B$. Our data structure is not limited to Euclidean space; its definition generalizes to all possible choices of query objects, data objects, and distance functions.

## 1 Introduction

Nearest-neighbor search is the problem of building a data structure for a given set of data points such that for any given query point we can efficiently determine a data point that has the smallest distance. Nearest-neighbor search is a fundamental problem in applications such as pattern recognition [9], data mining [11], content-based image retrieval [21], medical image databases [15], geographic information systems [22], data compression [3], and shape recognition [23].

Most nearest-neighbors research is focused on the Euclidean case. In the Euclidean case the data points and the query points are members of $\mathbb{R}^d$, and the distance is the Euclidean metric. If the dimension $d$ is small, then the query time and the storage space are usually studied as functions of $n$. For higher dimensions $d$ it is more appropriate to study the query time and the storage space as functions of *both* the number $n$ of data points and the dimension $d$. The

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 440–454, 2003.

objective is finding a data structure for higher dimensions that answers nearest-neighbor queries significantly faster than a linear search would. If unlimited storage space is available for the data structure, then for all possible sets of $n$ data points it is possible to build a data structure whose query time is only logarithmic in $n$. However, if only $O(n^c)$ storage space is available, for some constant $c$, then the best currently known data structures [1] achieve a worst-case query time of $\Omega(n^{1-c/\lceil d/2 \rceil})$, which is close to $\Omega(n)$ for higher dimensions $d$. The challenge of the nearest-neighbor problem is minimizing the query time given a limited amount of storage space. The query times of two different data structures should be compared only if the storage space of both data structures is bounded similarly.

### 1.1 Previous Results

Algorithms researchers have a tradition of studying the *worst-case* behavior of data structures and algorithms. The following worst-case upper bounds are known for nearest-neighbor search. Using randomized point-location structures for triangulated Voronoi diagrams [7,17] we can answer queries in $O(d^3 \log n)$ time using $O(n^{\lceil d/2 \rceil + \delta})$ storage space, for each fixed $\delta > 0$. Agarwal and Erickson [1] derive the following time–space tradeoff by combining existing data structures: If only $O(m)$ storage space is available, then nearest neighbors can be found in $n/m^{1/\lceil d/2 \rceil} \log^{O(1)} n$ time. These worst-case bounds are rather high; even in dimensions as low as three we can guarantee a polylogarithmic worst-case query time only if more than $\Theta(n^2)$ storage space is available.

In recent years, the attention of researchers has been shifting towards $(1+\epsilon)$-*approximate* nearest-neighbor search. Here the problem is to find a data point whose distance from the query point is within a factor $1+\epsilon$ of the actual nearest neighbor. The query time and the storage space are studied as functions of three parameters: the input size $n$, the dimension $d$, and the accuracy $\epsilon$. If the parameter $\epsilon > 0$ is fixed, then the asymptotic bounds for $(1+\epsilon)$-approximate nearest-neighbor are lower than the corresponding bounds that are known for nearest-neighbor search.

We mention worst-case upper bounds for $(1+\epsilon)$-approximate nearest-neighbor search. Arya et al. [4] have introduced the bounded box decomposition (BBD) tree, which is basically a KD tree that has been extended with an additional type of node. The construction of the BBD tree is independent of the approximation parameter $\epsilon$, and a single BBD tree can be used to perform queries for all $\epsilon \geq 0$. The BBD tree achieves a query time of $O(d(d/\epsilon)^d \log n)$ using $O(dn)$ storage space. Indyk and Motwani [14] have reduced finding $(1+\epsilon)$-approximate nearest neighbors to approximate point location in collections of balls. This results in a query time of $d \log^{O(1)} n$ at the cost of $O(1/\epsilon)^d n \log^{O(1)} n$ storage space. Kushilevitz, Ostrovsky, and Rabani [16] have reduced Euclidean nearest-neighbor search to several nearest-neighbor searches in a Hamming space. Their methods achieve a query time of $d^2 \log^{O(1)} n/\epsilon^2$ at the cost of $(dn)^{O(1/\epsilon^2)}$ storage space. Har-Peled [13] defines a space decomposition that approximates the Voronoi diagram of the data set. This decomposition gives rise to a compressed

quadtree of size $O(n\frac{\log n}{\epsilon^d}\log\frac{n}{\epsilon})$ that answers $(1+\epsilon)$-nearest neighbor queries in $O(\log(n/\epsilon))$ time. Chan [6] stores the data points in various ordered sequences, each of which is ordered by the shuffle order on the binary representations of the data points. This gives a query time of $O(\log n + (1/\epsilon)^d)$ using $O(dn)$ storage space, assuming that the model of computation allows the operation of bitwise exclusive-or on the binary representations.

In this paper, we revisit the exact nearest-neighbor problem. We perform average-case analysis instead of worst-case analysis. The data set is seen as a sequence of $n$ vectors that have been chosen independently and uniformly at random from some finite-volume subset $S$ of $\mathbb{R}^d$. The query time and the storage space are considered to be random variables. The objective is bounding the expectations of these random variables. In most average-case analyses, $S$ is the unit cube $[0,1]^d$ or the unit-radius ball $\mathbb{B}^d$.

We mention the previously known average-case bounds for exact nearest-neighbor search. Alt and Heinrich–Litan [2] describe a method that uses $dn$ storage and which achieves a query time of $O(nd/\ln n)$ in the average case for the unit cube. This result holds only for the $L_\infty$ metric; it does not apply to the Euclidean case. It is possible to reanalyze the Clarkson–Meiser data structure [7, 17] — which has the best known worst-case bounds — for the average case for random points in the unit ball. We find that the storage space is at least linear in the full-dimensional complexity of the Delaunay dual of the data set. Dwyer [10] has shown that the expectation of this complexity is $d^{\Theta(d)}n$. This means that the expected storage space of the Clarkson–Meiser data structure grows more than exponentially in the dimension $d$. Our data structure improves this storage-space bound, whereas the query time is kept polylogarithmic in $n$.

Up to here we only mentioned work that contains the best known asymptotic bounds for nearest-neighbor search. There is a huge volume of nearest-neighbor work that is not aimed solely at deriving asymptotic bounds, but which characterizes the performance of practical data structures [20,12].

### 1.2 Our Results

We propose a simple, practical data structure for exact nearest neighbor search. This data structure is a labeled directed acyclic graph in which each node has at most two outgoing arcs. We perform an average-case analysis for random points in the unit ball. We find that the average-case query time is $O(d^2 2^d \log^2 n)$. We derive two upper bounds for the average-case storage space. The first bound is $O(\gamma^d n)$, where $\gamma$ is an unknown value in the interval $[\sqrt{2}, 4\sqrt{2e}]$. The second bound is $O(d2^d n \log n)$. There is also a trivial worst-case limit for the storage space: By definition our data structure has at most $n(n+1)/2$ nodes. Furthermore, we show that our data structure can be used efficiently in secondary memory. We assume a simple, standard secondary-memory model in which blocks of size $B$ are transferred between secondary memory and main memory. In this model, the average query time is $O((d2^d \log^2 n)/B + \log n)$, and the average storage space is both in $O(\gamma^d N)$ and in $O(d2^d N \log n)$, where $N = n/B$.

## 2 Motivation

Most of the recent theoretical work on nearest neighbors focuses on finding worst-case bounds for $(1+\epsilon)$-approximation. In contrast, we study average-case bounds for the exact problem in this paper. Before we give our main results, we explain the advantages of our approach. In Section 2.1 we justify our choice of analyzing the average case over uniformly distributed data points. In Section 2.2 we explain why we think that our bounds are better than the known bounds for $(1+\epsilon)$-approximation.

### 2.1 Meaningfulness of the Average Case

It is widely agreed that the known worst-case bounds for the exact problem are too pessimistic; it is hard to find realistic inputs that approach the worst-case behavior. Average-case analysis is less pessimistic than worst-case analysis. However, we have to assure ourselves that the nearest-neighbor problem is meaningful for the random inputs.

In our analysis we assume that the input has is distributed uniformly in the $d$-dimensional unit ball $\mathbb{B}^d$. This means that each of the $n$ data points is chosen independently and uniformly from $\mathbb{B}^d$. Beyer et al. [5] have argued that nearest neighbors become meaningless in this model when the dimension $d$ approaches infinity while $n$ is kept constant: For each fixed query point $q$ and each constant $\epsilon > 0$, the probability that each data point is a $(1+\epsilon)$-approximate nearest neighbor tends to 1 if the dimension $d$ goes to infinity. For independent random data points this statement is intuitive because the distance to each data point is the square root of a sum of $d$ independent random variables.

In this paper we focus on the opposite case in which the dimension is small relative to the input size. Let $q$ be a fixed query point in $\mathbb{B}^d$. Suppose that the dimension $d$ is a constant. We keep adding random data points, and while doing that we maintain the shell that consists of all data points whose distance to $q$ is between 1 and $(1+\epsilon)$ times the distance of $q$ to its nearest neighbor. If we keep increasing the number of random data points, then the fraction of data points that lie in the shell goes to zero. Thus, if the dimension is small in comparison the number of random data points, then nearest neighbor *are* meaningful.

The data structure that we present in this paper beats a linear search only when the dimension $d$ is at most logarithmic in the input size $n$. This is not much of a restriction; it does not make much sense to do nearest neighbor search for vectors of dimension more than $\Theta(\log n)$. This is clearly illustrated by the Johnson–Lindenstrauss Lemma, which states that *no matter how large the dimension $d$ is* we can map all the $n$ points into a space of dimension $O(\log n/\epsilon^2)$ such that the distance between any two data points changes by only a factor of $1 \pm \epsilon$. Dasgupta and Gupta [8] have found a simple proof of the Johnson–Lindenstrauss lemma.

### 2.2 Exact versus Approximate

It is not fair to compare exact algorithms and $(1+\epsilon)$-approximation algorithms for the simple reason that $(1+\epsilon)$-approximation is easier than finding exact nearest neighbors. But suppose that we want to make some kind of comparison. The asymptotic bounds for $(1+\epsilon)$-approximation appear to be better than the corresponding bounds for exact computation if the parameter $\epsilon > 0$ is fixed. However, the following example suggests otherwise.

The most space-efficient $(1+\epsilon)$-approximation data structure has a query time of $O(\log n + (1/\epsilon)^d)$. For simplicity, let us assume that this data structure answers all $(1+\epsilon)$-approximate queries using at most $\log n + (1/\epsilon)^d$ distance computations. We want to beat linear search. This means that the number of distance computations has to be at most $n$. Now consider a database that has $10^9$ records, which are represented by feature vectors in $\mathbb{R}^d$. If we want to find 1.01-approximate nearest neighbors, then the data structure beats linear search only for dimensions 4 and smaller. In contrast, we can guarantee that our own data structure finds *exact* nearest neighbors faster than a linear search for at least 13 dimensions, in the average case, for this example. (In section 4.2 we prove an upper bound on the average-case number of distance computations that is roughly $2e^2(d+3)2^d \ln^2 n$.)

## 3 Our Data Structure

Mulmuley [19] uses the term history graph to describe the online point-location data structure that is a by-product of the randomized incremental construction of a geometric structure. Our data structure — the *Voronoi history graph* — is a history graph for Voronoi diagrams. Each node of the Voronoi history graph represents a Voronoi cell in some subset of the data set.

We define Voronoi cells in terms of halfspaces. The *closed halfspace* $H(r,p)$ is the set of all points $q \in \mathbb{R}^d$ for which $\|q-r\| \leq \|q-p\|$. We assume that a *data set* is a nonempty, finite subset of $\mathbb{R}^d$. The *closed Voronoi cell* $V_S(r)$ of the data point $r$ in the data set $S$ is the intersection of all closed halfspaces $H(r,p)$ for $p \in S \setminus r$. (We write $S \setminus r$ instead of $S \setminus \{r\}$.) The *Voronoi diagram* of the data set $S$ is the collection of all closed Voronoi cells $V_S(p)$ with $p \in S$. We can characterize nearest neighbors in terms of Voronoi cells: A data point $r \in S$ is a nearest neighbor of $q$ if and only if $q \in V_S(r)$.

The Voronoi history graph is a labeled directed acyclic graph. It consists of a set of *nodes* (vertices) and a set of *arcs* (directed edges). The outdegree of each node is either zero or two. Nodes with outdegree zero are called *terminal*, and nodes with outdegree two are called *binary*. Each node $v$ has an integer label $j_v$, which is called the *index* of $v$. Each arc is either *conservative* or *progressive*. We say that an arc is conservative if it connects two nodes with the same index; otherwise, we call the arc progressive. We say that a child $w$ of $v$ is conservative (progressive) if the arc $v \to w$ connecting $v$ and $w$ is conservative (progressive).

The Voronoi history graph is completely determined by an ordered sequence of data points $p_1, p_2, \ldots, p_n$. For each $1 \leq j \leq n$, we call $j$ the index of the data

point $p_j$. For each node $v$ we say that $p_{j_v}$ is the *associated data point* for $v$. For each $1 \leq i \leq n$, we denote the set of the first $i$ elements of the data-point sequence by $P^i$. We write $V_i(j)$ instead of $V_{P^i}(p_j)$, and we write $H(j_1, j_2)$ instead of $H(p_{j_1}, p_{j_2})$.

We define the Voronoi history graph for the data-point sequence $p_1, p_2, \ldots, p_n$ inductively. The graph for $p_1$ has one terminal node with index 1 and no arcs. We call this node the *root node*. Given the graph for $p_1, p_2, \ldots, p_i$, we obtain the graph for $p_1, p_2, \ldots, p_{i+1}$ by adding new nodes and arcs as follows. We add a terminal node $x$ with index $i + 1$. For each terminal node $v$ that satisfies $V_i(j_v) \neq V_{i+1}(j_v)$ we add the following: a terminal node $w$ with the same index as $v$, the conservative arc $v \to w$, and the progressive arc $v \to x$.

We use a recursive query procedure to answer nearest-neighbor queries. The procedure follows a path from the root node to the terminal node whose associated data point is the nearest neighbor. The query procedure has two parameters: a node $v$ and a query point $q$. In the initial call of the query procedure we use the root node $v$ and the query point $q$ as parameters. The query procedure works as follows. If $v$ is a terminal node, then the procedure returns the associated data point of $v$. If $v$ is a binary node, then we make a case distinction. Let $w$ be the conservative child of $v$, and let $x$ be the progressive child of $v$. In the case that $q$ lies in $H(j_w, j_x)$ the procedure calls itself with parameters $w$ and $q$. Otherwise, the procedure calls itself with parameters $x$ and $q$.

## 4 An Average-Case Analysis

The existing techniques for analyzing history graphs cannot be used to show that the Voronoi history graph is efficient. The reason for this is that Voronoi diagrams do not satisfy the *bounded degree property*: The number of points that define a Voronoi cell is not bounded above by a constant. (The existing techniques *do* work for history graphs of triangulated Voronoi diagrams, but in Section 1 we saw that these data structures require significantly more storage space.) In this section, we prove bounds for the Voronoi history graph using combinatorial properties of Delaunay graphs of uniformly distributed random point sets.

Our analysis of the Voronoi history graph is structured as follows. In Section 4.1 we prepare ourselves for the actual analysis by identifying each node with an integer pair. We bound the query time in Section 4.2. After that, we derive multiple bounds for the storage space in Section 4.3. Finally, in Section 4.4, we translate our time–space bounds to a standard secondary-memory model. In our analysis we repeatedly use lemmas that concern Delaunay graphs of random point sets; to keep the analysis clean, we give these lemmas and their proofs separately in Section 5.

In the analysis we assume that $p_1, p_2, \ldots, p_n$ is a random sequence of data points each of which has been chosen independently from a $d$-dimensional ball under the uniform distribution. We study the Voronoi history graph that is determined by this sequence. We treat the query time and the storage space as random variables.

### 4.1 A Representation for the Nodes

In our analysis we need to refer to specific nodes of the Voronoi history graph. The index of a node alone does not identify the node uniquely; several nodes may share the same index. Each node $v$ is identified with the pairs of integers $(j, i)$. We define $j$ as the index $j_v$. The integer $i$ – the *iteration* of $v$ – is defined next.

We specify the iteration of each node by giving an inductive definition of the Voronoi history graph in terms of the iteration-index pairs. This definition upholds the following invariant: The set of terminal nodes for the graph of $p_1, p_2, \ldots, p_i$ is the set of all integer-pairs $(j, i)$ with $1 \leq j \leq i$. The Voronoi history graph for the sequence $p_1$ has one node $(1, 1)$ and no arcs. Given the Voronoi history graph for $p_1, p_2, \ldots, p_i$, we obtain the Voronoi history graph for $p_1, p_2, \ldots, p_{i+1}$ as follows. We add the terminal node $(i + 1, i + 1)$. For each terminal node $(j, i)$ we make a case distinction. If $V_i(j) = V_{i+1}(j)$, then we rename $(j, i)$ to $(j, i + 1)$ and we replace each arc $v \to (j, i)$ by $v \to (j, i + 1)$. If $V_i(j) \neq V_{i+1}(j)$, then we add the terminal node $(j, i + 1)$, the conservative arc $(j, i) \to (j, i + 1)$, and the progressive arc $(j, i) \to (i + 1, i + 1)$.

### 4.2 The Query Time

We analyze the query time for a fixed query point $q$. We mention explicitly that the query point is *not* chosen randomly. We derive bounds that hold for each fixed choice of $q$ in $\mathbb{R}^d$.

The query time is a linear function of $d$ times the length of the *search path*. The search path is the set of all nodes that are passed as a parameter to the query procedure starting from the initial call. For each iteration $1 \leq i \leq n - 1$, we define $m(i)$ as the smallest index $1 \leq j \leq i$, satisfying $q \in V_i(j)$. Observe that the search path is the set of all integer-pairs $(m(i), i)$ with $1 \leq i \leq n$ that are nodes in the Voronoi history graph.

We proceed by studying the length $L$ of the search path. For each iteration $1 \leq i < n$, we define a random variable $L_i$ by setting $L_i = 1$ if the search path contains a node with iteration $i$, and setting $L_i = 0$ otherwise. Observe that $L_i = 1$ if and only if $(m(i), i)$ is a node in the Voronoi history graph. The search-path length can be expressed as $L = 1 + \sum L_i$, where the summation ranges over all $i \in \{ 1, 2, \ldots, n - 1 \}$.

We bound the expected search-path length by exploiting properties of the *Delaunay graph* of the data points. We define Delaunay graphs in terms of Voronoi cells. We say that two elements $r$ and $p$ of a data set $S$ are *Delaunay neighbors* if $V_{S \setminus p}(r) \neq V_S(r)$. Delaunay neighborship is a symmetric relation. The Delaunay graph of a data set $S$ is the undirected graph whose vertex set equals $S$ and whose edge set consists of all unordered pairs from $S$ that are Delaunay neighbors relative to $S$. Observe that $(j, i)$ is a node in the Voronoi history graph if and only if there is and edge between $p_j$ and $p_{i+1}$ in the Delaunay graph of $P^{i+1}$.

We continue by studying the probability of $L_i = 1$. For $1 \leq j \leq k < n$, define the random variable $D_j^k$ as the number of Delaunay neighbors of $p_j$ in the set

$P^k$. Let $\delta > 0$ be some constant, and assume that $i \geq 2d/\delta$. Let the function $\tau$ by defined by applying Lemma 3 with $\alpha = 2$. The probability that there exists some $1 \leq j \leq i+1$ for which $D_j^{i+1} > \tau(d,n)$ is at most $1/n$. It follows that

$$\mathbf{Pr}[\, L_i = 1 \,] \leq \frac{1}{n} + \frac{1}{i}\tau(d,n) \ . \tag{1}$$

Using linearity of expectation we bound above the total search-path length as

$$\begin{aligned} \mathbf{E}[\, L \,] &\leq 1 + 2d/\delta + \sum \frac{1}{n} + \frac{1}{i}\tau(d,n) \\ &\leq 2 + 2d/\delta + \tau(d,n) \ln n. \end{aligned} \tag{2}$$

Using that the expected query time is linear in $d$ times the expected search-path length we obtain the following result:

**Theorem 1.** *Suppose that $p_1, p_2, \ldots, p_n$ are random points that have been chosen independently and uniformly from the unit ball $\mathbb{B}^d$. Then, using the Voronoi history graph, the nearest neighbor of each query point $q \in \mathbb{R}^d$ can be found in $O(d^2 2^d \log^2 n)$ expected time.*

### 4.3 The Storage Space

The Voronoi history graph itself uses storage space linear in the number of nodes. Remember that the size of each node is independent of $d$; each node stores only the index of a data point and two pointers. The data points themselves are stored in a separate array, which takes up $\Theta(dn)$ additional space.

We study the number of nodes in the Voronoi history graph as a random variable $X$. Recall that for $j \leq i < n$, the pair $(j,i)$ is a node of the Voronoi history graph if and only if there is an edge between $p_j$ and $p_{i+1}$ in the Delaunay graph of $P^{i+1}$. It follows that the number of nodes with iteration $i$ equals $D_{i+1}^{i+1}$. This means that $X$ equals $n + \sum D_k^k$, where the summation ranges over all $k \in \{\, 2, 3, \ldots, n \,\}$. Using linearity of expectation we find that

$$\mathbf{E}[\, X \,] = n + \sum \mathbf{E}\big[\, D_k^k \,\big] \ . \tag{3}$$

We use this equation to derive two distinct upper bounds for $\mathbf{E}[\, X \,]$.

The first bound has a linear dependence on $n$. From Lemma 4 we learn that the expectation of $D_k^k$ equals $\gamma^d$ for some $\gamma \in [\sqrt{2}, 4\sqrt{2e}]$. Substituting this in 3 we find that

$$\mathbf{E}[\, X \,] \leq \gamma^d n \ . \tag{4}$$

The second bound has a better guarantee for the dependence on $d$ at the cost of an extra factor that is logarithmic in $n$. We apply Lemma 3 choosing $\alpha = 1$. For $k \geq 2d/\delta$, the probability of $D_k^k > \tau(d,n)$ is at most $1/n$. It follows that

$$\mathbf{E}\big[\, D_k^k \,\big] \leq 1 + \tau(d,n) \ . \tag{5}$$

Substituting this inequality in (3) we find the bound

$$\mathbf{E}[\, X \,] \leq (2d/\delta)^2 + 2n + \tau(d,n)n \ . \tag{6}$$

From (4) and (6) we obtain asymptotic bounds for the expected storage space:

**Theorem 2.** *Suppose that $p_1, p_2, \ldots, p_n$ are random points that have been chosen independently and uniformly from the unit ball $\mathbb{B}^d$. Then the Voronoi history graph has at most $\binom{n}{2}$ nodes, and the expected storage space is both in $O(\gamma^d n)$ and in $O(d2^d n \log n)$.*

### 4.4 Secondary Memory

For very large data sets the Voronoi history graph does not fit into main memory. In such cases the data structure has to be stored in secondary memory. Our query algorithm would have to transfer any data it needs from secondary memory to primary memory. The problem is that these transfers are computationally expensive.

We analyze the efficiency of the Voronoi history graph using a simple, standard *secondary-memory model*. We assume that data is stored in secondary memory in blocks of size $B$. The block size is treated seen as a parameter just like $d$ and $n$. We define $N = n/B$. We assume that each access to secondary memory transfers an entire block; it is not possible to access data within a block in secondary memory without transferring the entire block into main memory. We count each block transfer as a single I/O operation. In the secondary-memory model we define the query time and the storage space as follows: The query time equals the number of I/O operations that are needed to answer a query. The storage space equals the number of blocks that are needed to store the data structure.

We show that the Voronoi history graph can be embedded efficiently into secondary memory. We conveniently assume that each block is able to store exactly $B$ nodes. This assumption does not influence our asymptotic bounds because each node occupies a small, constant amount of storage, which is independent of $d$ and $n$. We define the embedding as follows. First, we put the nodes of the Voronoi history graph into an ordered sequence in which $(i_1, j_1)$ precedes $(i_2, j_2)$ if and only if $i_1 \leq i_2$ or $i_1 = i_2$ and $j_1 \leq j_2$. We store the nodes in secondary memory by putting the first $B$ nodes in the first block, the second $B$ nodes in the second block, etc. The last block may not be completely filled.

Observe that the (secondary-memory) query time is linear in $\lceil L/B \rceil + J$, where $L$ is the search-path length, and $J$ is the number of progressive arcs on the search path. For each $2 \leq k \leq n$ we define a $\{\, 0, 1 \,\}$-valued random variable $J_k$ which equals 1 if and only if $(k, k)$ is a node on the search path. Clearly, $J = \sum J_k$, where $k$ ranges over $2, 3, \ldots, n$. If $(k, k)$ is on the search path, then the query point lies in the Voronoi cell $V_k(k)$. The probability of the latter event is $1/k$. Using linearity of expectation we find that the expectation of $J$ is $O(\log n)$. Our observations lead to the following bounds for the Voronoi history graph in the secondary-memory model:

**Theorem 3.** *Consider the Voronoi history graph for random points $p_1, p_2, \ldots, p_n$ that have been chosen independently and uniformly from the unit ball $\mathbb{B}^d$. In the secondary-memory model, the expected query time is $O((d2^d \log^2 n)/B + \log n)$, for each query point, and the expected storage space is $O(\gamma^d N)$ and $O(d2^d N \log n)$*

## 5 Delaunay Graphs for Random Point Sets

Our analysis of the Voronoi history graph is based on combinatorial properties of Delaunay graphs of random point sets. To be precise, the analysis makes use of bounds for the degree of a point in the Delaunay graph. In Section 5.1 we prove high-probability upper bounds for the degree of a point. In Section 5.2 we find an upper bound for the expected degree of a point by applying Dwyer's bounds [10] for the expected number of edges.

### 5.1 High-Probability Bounds for the Degree

The degree of a point in the Delaunay graph equals the number of Delaunay neighbors of that point. We characterize Delaunay neighborship using candidate regions. Candidate regions were defined earlier (in a slightly different way) by Clarkson [7]. We define the *candidate region* $C_S(r)$ of a data point $r$ and a data set $S$ as the set of all $x \in \mathbb{R}^d$ for which $V_S(r) \neq V_{S \cup \{x\}}(r)$. Observe that $r \in S$ and $p \notin S$ are Delaunay neighbors in $S \cup \{p\}$ if and only if $p$ is contained in the candidate region $C_S(r)$.

We can equivalently define candidate regions as a unions of open balls. We define the *open ball* $\mathrm{B}(c, \rho)$ with center $c \in \mathbb{R}^d$ and radius $\rho \in \mathbb{R}$ as the set of all $x \in \mathbb{R}^d$ for which $\|x - c\| < \rho$. The boundary of $\mathrm{B}(c, \rho)$ is the set of $x \in \mathbb{R}^d$ for which $\|x - c\| = \rho$. The unit ball $\mathbb{B}^d$ is the open ball $\mathrm{B}(o, 1)$ that is centered at the origin $o$. The candidate region $C_S(r)$ is the union of all open balls whose boundaries contain $r$ and which do not contain any point of $S$. Observe that $S' \subseteq S$ implies $C_S(r) \subseteq C_{S'}(r)$.

We define the *relative volume* of a subset $U$ of $\mathbb{B}^d$ as the volume of $U$ divided by the volume of $\mathbb{B}^d$. The relative volume of $U$ equals the probability that a uniformly chosen random point from $\mathbb{B}^d$ lies in $U$.

**Lemma 1.** *Let $r$ be some point of $\mathbb{B}^d$, and let $S$ be a finite set of points in $\mathbb{B}^d$. If the intersection of the candidate region $C_S(r)$ with $\mathbb{B}^d$ has volume greater than $\mu$, then there exists a open ball $B$ whose boundary contains $r$, that contains no point of $S$, and whose intersection with $\mathbb{B}^d$ has a volume greater than $(1/2)^d \mu$.*

*Proof.* Let $\mathcal{B}$ be the collection of all open balls that contain in $r$ in the boundary and that contain no point of $S$. Recall that $C_S(r)$ equals the union of $\mathcal{B}$. We actually need to prove the following: If for all balls in $\mathcal{B}$, the intersection with $\mathbb{B}^d$ has volume at most $(1/2)^d \mu$, then the intersection of $\bigcup \mathcal{B}$ with $\mathbb{B}^d$ has volume at most $\mu$.

Consider the case that all elements of $\mathcal{B}$ are contained in $\mathbb{B}^d$. In this case it is obvious that the union of $\mathcal{B}$ has at most volume $\mu$; the union of all balls with volume $(1/2)^d\mu$ that contain $r$ in the boundary is itself a ball with volume $\mu$. For the case that not all $\mathcal{B}$ are contained in $\mathbb{B}^d$, the volume of the union of $\mathcal{B}$ is less than $2^d$ times the volume of the largest intersection of an element of $\mathcal{B}$ with $\mathbb{B}^d$. □

**Lemma 2.** *Let $r$ be an element of $\mathbb{B}^d$. Suppose that $S$ is the set of elements in a sequence of $m$ random points that have been chosen independently and uniformly from $\mathbb{B}^d$. The probability that the intersection of $C_S(r)$ with $\mathbb{B}^d$ has relative volume greater than $\mu$ is less than or equal to*

$$\exp\left(d\ln(em/d) - 2^{-d}(m-d)\mu\right) \ . \tag{7}$$

*Proof.* Suppose that the intersection of $C_S(r)$ with $\mathbb{B}^d$ has relative volume greater than $\mu$. Then, by Lemma 1 there exists some ball $B$ whose boundary contains $r$, that does not contain any point of $S$, and whose intersection with $\mathbb{B}^d$ has a relative volume greater than $(1/2)^d\mu$. The ball $B$ is contained in a larger ball $B'$ that contains no point of $S$ and whose boundary contains $r$ and $d$ distinct points of $S$. Together with $r$ these $d$ points of $S$ determine the ball $B'$ uniquely. Thus there is a subset of $d$ points of $S$ that determine a ball $B'$ with relative volume greater than $(1/2)^d\mu$ that does not contain any point $S$. The probability that a fixed tuple of $d$ random points determines such a ball is less than or equal to $\left(1-2^{-d}\mu\right)^{m-d}$. This probability is bounded above by $\exp\left(-2^{-d}\,(m-d)\,\mu\right)$. By summing this upper bound over all $\binom{m}{d} \leq (em/d)^d$ choices of $d$ points from $S$ we find the upper bound claimed in the lemma. □

Consider a random sequence of data points $p_1, p_2, \ldots, p_n$ that have been chosen independently and uniformly at random from the unit ball $\mathbb{B}^d$. Without loss of generality we may assume that no two points in this sequence are the same, since the probability of two points being the same is zero. Recall that the random variable $D_j^i$ is defined as the number of Delaunay neighbors of $p_j$ in the set $P^i$.

**Lemma 3.** *Let $\delta > 0$ and $\alpha > 0$ be constants. Define*

$$\tau(d,n) \stackrel{def}{=} \frac{\alpha+d+1}{1-\delta} \cdot 2e^2 2^d \ln n \ . \tag{8}$$

*Then for all $n \geq 4e$, $2d/\delta \leq k \leq n$, and $1 \leq j \leq k$,*

$$\mathbf{Pr}\left[\, D_j^k > \tau(d,n) \,\right] < n^{-\alpha} \ . \tag{9}$$

*Proof.* The random variables $D_j^k$ with $1 \leq j < k$ have identical distributions, so we continue by studying $D_k^k$. We prove something stronger than required: We show that the claimed tail bound (3) holds for all choices $r$ of $p_k$. We assume that $p_k$ is not random but fixed to some $r \in \mathbb{R}^d$.

The random variable $D_k^k$ equals the number of Delaunay neighbors of $r$ in the set $P^k$. The set of Delaunay neighbors of $r$ in $P^k$ is the set of all points $p_j$ with $1 \le j < i$ that satisfy $p_j \in C_{P^k \setminus p_j}(r)$.

For our convenience we assume that $k$ is odd; it is easy to generalize to even $k$. We subdivide $P^{k-1}$ into two balanced subsets $A$ and $B$, which are both of size $m \stackrel{\text{def}}{=} (k-1)/2$. We define $A$ as the set $\{ p_1, p_2, \ldots, p_m \}$, and we define $B$ as the set $\{ p_m, p_{m+1}, \ldots, p_{k-1} \}$. We bound above $D_k^k$ by a sum of $\{0,1\}$-valued random variables $N_j$ with $1 \le j \le k-1$. For $j \le m$ we set $N_j = 1$ if and only if $p_j$ is contained in $C_B(r)$. For $j > m$ we set $N_j = 1$ if and only if $p_j$ is contained in $C_A(r)$. Define $U = N_1 + N_2 + \cdots + N_m$ and $V = N_{m+1} + N_{h+2} + \cdots + N_{k-1}$, The degree $D_k^k$ is bounded above by the sum of $U$ and $V$. This implies that

$$\mathbf{Pr}\big[\, D_k^k > \tau \,\big] \le \mathbf{Pr}[\, U > \tau/2 \,] + \mathbf{Pr}[\, V > \tau/2 \,]. \tag{10}$$

We continue by bounding above the probability of $U > \tau/2$. (The probability of $V > \tau/2$ can be bounded analogously.) Define the random variable $R$ as the relative volume of the intersection of $C_B(r)$ with $\mathbb{B}^d$. Now

$$\mathbf{Pr}[\, U > \tau/2 \,] \le \mathbf{Pr}[\, R > \mu \,] + \mathbf{Pr}[\, U > \tau/2 \mid R \le \mu \,]. \tag{11}$$

We bound above the two probabilities on the right-hand side of this inequality.

The probability of $U > \tau/2$ is given by Equation (7) of Lemma 2. We continue by bounding the probability of $U > \tau/2$ under the condition that $R \le \mu$. The expectation of $U$ given that $R \le \mu$ is at most $\mu m$. Observe that the random variables $N_1, N_2, \ldots, N_m$ of which $U$ is the sum are mutually independent. This means we may use the following Chernoff bound: The conditional probability that $U$ exceeds $c > 1$ times its conditional expectation given that $R \ge \mu$ is at most $(e/c)^{c\mu m}$. This is an application of the tail bound given by Motwani and Raghavan [18] in Theorem 4.1. Substituting $c = \tau/(2\mu m)$, we find that

$$\mathbf{Pr}[\, U > \tau/2 \mid R \le \mu \,] < (2e\mu m/\tau)^{\tau/2}. \tag{12}$$

Substituting the inequalities (7) and (12) in the right-hand side of inequality (11), we obtain the bound

$$\mathbf{Pr}[\, U > \tau/2 \,] \le \exp\big(d \ln(em/d) - 2^{-d}(m-d)\mu\big) + (2e\mu m/\tau)^{\tau/2}. \tag{13}$$

We want to bound above both summands on the right-hand side of this inequality by $\frac{1}{4} n^{-\alpha} = \exp(-\ln 4 - \alpha \ln n)$. To bound above the first right-hand term of (13), it is sufficient to choose

$$\mu \stackrel{\text{def}}{=} \frac{\alpha + d + 1}{1 - \delta} \cdot 2^d \frac{\ln n}{m}. \tag{14}$$

To bound above the second right-hand term of (13), it is sufficient to choose $\tau = \tau(d,n)$ as given in (8). □

### 5.2 The Expected Degree

The following bound for the expected degree of a point is a consequence of results by Dwyer [10].

**Lemma 4.** *There exists a constant* $\gamma \in [\sqrt{2}, 4\sqrt{2e}]$ *such that the expectation of* $D_j^k$ *is at most* $\gamma^d$ *for all* $1 \leq k \leq n$ *and all* $1 \leq j \leq k$.

*Proof.* Dwyer [10] has proved that the expected number of edges in the Delaunay graph for $P^k$ is at most $\gamma^d k$, where $\gamma$ is an unknown constant, which must lie in the interval $[\sqrt{2}, 4\sqrt{2e}]$. Observe that the expected number of edges for $P^k$ is the sum of $\mathbf{E}[\, D_j^k \,]$ over $1 \leq j \leq k$. The result follows because all the random variables $D_j^k$ with $1 \leq j \leq k$ have identical distributions. □

## 6 Conclusions

We have proposed the Voronoi history graph as a data structure for nearest neighbor search in higher-dimensional spaces. We have analyzed this data structure for the average case over independent, uniformly distributed random points in a $d$-dimensional ball. For fixed dimensions, the Voronoi history graph achieves a query time of $O(\log^2 n)$ using only $O(n)$ storage space. If we consider variable dimensions, then both the query time and the storage space have a dimension-dependent factor that depends exponentially on $d$. This is an improvement over previously known bounds for exact nearest-neighbor search; these bounds all have a super-exponential factor like $d^{\Theta(d)}$ either in the query time (e.g., BBD trees) or in the storage space (e.g., history graphs of triangulated Voronoi diagrams).

Besides good upper bounds, the Voronoi history graph has advantages such as simplicity and ease of implementation. The data structure itself is simple — a collection of nodes. Each node stores only the index of a data point and two pointers to other nodes. Our query algorithm does not require any extra storage: It simply walks through the graph directly to the terminal node whose associated data point is the nearest neighbor. In each node the query procedure computes the inner product of two vectors to determine which node has to be visited next.

The Voronoi history graph can be stored efficiently in secondary memory. The distribution of the nodes into blocks is straightforward. The given time–space bounds for the real RAM model translate into efficient bounds for a secondary memory model. In this model, the average query time is $O((d2^d \log^2 n)/B + \log n)$, and the average storage space is both in $O(\gamma^d N)$ and in $O(d2^d N \log n)$, where $B$ is the block size and $N = n/B$.

The Voronoi history graph can be generalized for all possible types of data objects and choices of distance functions: Any generalization requires only a redefinition of the notion of a halfspace. To implement the data structure for a generalized setting two problems have to be addressed. The first problem is computing distances between the objects. The second problem is testing whether adding a new data point changes a Voronoi cell.

## References


1. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, 1999.
2. H. Alt and L. Heinrich-Litan. Exact $L_\infty$ nearest neighbor search in high dimensions. In *Proceedings of the 17th ACM Symposium on Computational Geometry*, pages 157–163, 2001.
3. S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Proceedings of the 1993 IEEE Data Compression Conference*, pages 381–390, 1993.
4. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45:891–923, 1998.
5. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "Nearest Neighbor" meaningful. In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235, 1999.
6. T. M. Chan. Closest-point problems simplified on the RAM. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.
7. K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.
8. S. Dasgupta and A. Gupta. An elementary proof of the Johnson–Lindenstrauss lemma. Technical Report TR-99-006, International Computer Science Institute, 1999.
9. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification.* Wiley, 2000.
10. R. A. Dwyer. The expected number of $k$-faces of a Voronoi diagram. *Computers and Mathematics with Applications*, 26(5):13–19, 1993.
11. C. Faloutsos and K.-I. Lin. *FastMap:* a fast algorithm for indexing, data-mining, and visualization of traditional and multimedia databases. In *Proceedings of the 1995 ACM–SIGMOD International Conference on Management of Data*, pages 163–173, 1995.
12. J. Goldstein and R. Ramakrishnan. Contrast plots and p-sphere trees: Space vs. time in nearest neighbor searches. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 429–440, 2000.
13. S. Har-Peled. A replacement for voronoi diagrams of near linear size. In *Proceedings of the 42th IEEE Symposium on the Foundations of Computer Science*, pages 94–103, 2001.
14. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 604–613, 1998.
15. F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *The International Journal on Very Large Data Bases*, pages 215–226, 1996.
16. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 614–623, 1998.
17. S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.
18. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University, 1995.

19. K. Mulmuley. *Computational Geometry: An introduction through randomized algorithms*. Prentice Hall, 1994.
20. B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proceedings of the 16th International Conference on Data Engineering*, pages 589–598, 2000.
21. A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: tools for content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, 1996.
22. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison–Wesley, 1990.
23. J. Vleugels and R. C. Veltkamp. Efficient image retrieval through vantage objects. In *Proceedings of the 3rd International Conference on Visual Information Systems*, pages 575–584, 1999.

# Author Index