Lecture Notes in Computer Science 2245

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

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

Ramesh Hariharan Madhavan Mukund
V. Vinay (Eds.)

FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science

21st Conference
Bangalore, India, December 13-15, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Ramesh Hariharan
V. Vinay
Indian Institute of Science, Department of Computer Science and Automation
Bangalore 560012, India
E-mail: {ramesh/vinay}@csa.iisc.ernet.in

Madhavan Mukund
Chennai Mathematical Institute
92 G.N. Chetty Road, Chennai 600 017, India
E-mail: madhavan@cmi.ac.in

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Foundations of software technology and theoretical computer science :
21th conference ; proceedings / FST TCS 2001, Bangalore, India, December
13 - 15, 2001. Ramesh Hariharan ... (ed.). - Berlin ; Heidelberg ; New York ;
Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2245)
ISBN 3-540-43002-4

CR Subject Classification (1998): F.3, D.3, F.4, F.2, F.1, G.2

ISSN 0302-9743
ISBN 3-540-43002-4 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

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

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10845907 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 21st international conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2001), organized under the auspices of the Indian Association for Research in Computing Science (IARCS).

This year's conference attracted 73 submissions from 20 countries. Each submission was reviewed by at least three independent referees. In a departure from previous conferences, the final selection of the papers making up the program was done through an electronic discussion spanning two weeks, without a physical meeting of the Program Committee (PC).

Since the PC of FSTTCS is distributed across the globe, it is very difficult to fix a meeting whose time and venue is convenient for a substantial fraction of the PC. Given this, it was felt that an electronic discussion would enable all members to participate on a more equal footing in the final selection. All reviews, scores, and comments were posted on a secure website, with a mechanism for making updates and automatically sending notifications by email to relevant members of the PC. All PC members participated actively in the discussion. The general feedback on the arrangement was very positive, so we hope to continue this in future years.

We had five invited speakers this year: Eric Allender, Sanjeev Arora, David Harel, Colin Stirling, and Uri Zwick. We thank them for having readily accepted our invitation to talk at the conference and for providing abstracts (and even full papers) for the proceedings.

Two satellite workshops were organized in conjunction with FSTTCS this year. The conference was preceded by a two-day workshop (December 11–12) on Quantum Computing in Bangalore, organized by Harry Buhrman and Umesh Vazirani. Following the conference, on December 17–18, there was a two-day workshop on Reasoning about Large and Infinite State Systems in the nearby city of Chennai (Madras), organized by P.S. Thiagarajan.

We thank all the reviewers and PC members, without whose dedicated effort the conference would not have been possible. We thank the Organizing Committee for making the arrangements for the conference. As usual, Alfred Hofmann and his team at Springer-Verlag were very helpful in preparing the proceedings.

December 2001

Ramesh Hariharan
Madhavan Mukund
V Vinay

Program Committee

S. Arun-Kumar (IIT, Delhi)
Julian Bradfield (Edinburgh)
Harry Buhrman (CWI)
Gautam Das (Microsoft Res)
Anuj Dawar (Cambridge)
Alan Frieze (CMU)
Naveen Garg (IIT, Delhi)
Ramesh Hariharan (IISc) (Co-chair)
Oded Maler (Verimag)
Peter Bro Miltersen (BRICS, Aarhus)
Madhavan Mukund (CMI) (Co-chair)
Paritosh Pandya (TIFR)
Sanjiva Prasad (IIT, Delhi)
Jaikumar Radhakrishnan (TIFR)
Uday Reddy (Birmingham)
Natarajan Shankar (SRI)
Aravind Srinivasan (Bell Labs)
Madhu Sudan (MIT)
Amnon Ta-Shma (Tel-Aviv)
V. Vinay (IISc) (Co-chair)
Pascal Weil (Bordeaux)
Thomas Wilke (Kiel)
D. Williamson (IBM, Almaden)

Organizing Committee

Vijay Chandru (IISc)
Ramesh Hariharan (IISc)
Swarup K. Mohalik (Sasken)
L. S. Ramakrishnan (String Labs)
V. M. S. Vasan (String Labs)
G. Venkatesh (Sasken)
V. Vinay (IISc)

Referees

Parosh Aziz Abdulla
Bharat Adsul
Klaus Aehlig
Rajeev Alur
V.S. Anil Kumar
A. Arnold
V. Arvind
Guillem Godoy Balil
Arye Barkan
Arnold Beckmann
Giampaolo Bella
Michael Benedikt
Saddek Bensalem
Yonatan Bilu
Avrim Blum
V. Borkar
Ahmed Bouajjani
Marius Bozga
Benoît Caillaud
Iliano Cervesato
Bernadette Charron-Bost
Samir Chopra
Benny Chor
Ken Clarkson
Federico Crazzolara
R. L. Crole
Paul Curzon
Silvano Dal-Zilio
Joerg Desel
Juergen Dingel
Deepak D'Souza
Irène Durand
Martin Dyer
Javier Esparza
Kousha Etessami
Mike Fellows
Wan Fokkink
Lance Fortnow
Carsten Fritz
Leslie Goldberg
Andy Gordon
Martin Grohe
Anupam Gupta
Peter Habermehl
Mitch Harris
Thomas Hildebrandt
Martin Hofmann
Purushothaman S. Iyer
R. Jagadeesan
Petr Jančar
Mark Jerrum
Haim Kaplan
Joost-Pieter Katoen
T. Kavitha
Astrid Kiehn
Felix Klaedtke
Josva Kleist
Jochen Konemann
Ralf Kuesters
Orna Kupferman
Kim G. Larsen
Salvatore La Torre
K. Lodaya
Satyanarayna V. Lokam
Ben Lukoschus
P. Madhusudan
Bruce Maggs
Meena Mahajan
Monika Maidl
Rupak Majumdar
Ken McMillan
Dale Miller
Mark Moir
Jean-Francois Monin
Yoram Moses
Leonardo de Moura
Anca Muscholl
Kedar S. Namjoshi
M. K. Narasimha
K. Narayan Kumar
Ilan Newman
Peter Niebert
Michael Norrish
Luke Ong
Itsik Pe'er
Laure Petrucci
Anna Philippou
Elon Portugaly

Alexander Rabinovich
Balaji Raghavachari
R. Ramanujam
Abhiram Ranade
R. Ravi
Laurent Regnier
Klaus Reinhard
Eike Ritter
Hein Roehrig
Harald Ruess
John Rushby
Norbert Schirmer
Alan Schmitt
Claus Schröter
Nicole Schweikardt
Helmut Schwichtenberg
Stefan Schwoon
Luc Segoufin
Géraud Sénizergues
Anil Seth
Peter Sewell
Roded Sharan
David Shmoys
Joseph Sifakis
Purnendu Sinha
D. Sivakumar
Sivaramakrishnan Sivasubramanian
Milind Sohoni
Ashok Sreenivas
Venkatesh Srinivasan
Alin Stefanescu
L.J. Steggles
Colin Stirling
Martin Strecker
Dave Stringer-Calvert
K.V. Subrahmanyam
S. Sudarshan
L. Sunil Chandran
Sridhar Tayur
P.S. Thiagarajan
John G. Thistle
Sophie Tison
Ashish Tiwari
Hans Toetenel
Howard Wong Toi
Stavros Tripakis
Tomas Uribe
Vincent Vanackere
Kasturi Varadarajan
Vasco T. Vasconcelos
Kevin Watkins
Ingo Wegener
David Williamson
Ronald de Wolf
Mike Wooldridge
Marc Zeitoun

Table of Contents

Invited Papers

When Worlds Collide:
Derandomization, Lower Bounds, and Kolmogorov Complexity 1
Eric Allender (Rutgers)

Approximation Schemes for Geometric NP-Hard Problems: A Survey 16
Sanjeev Arora (Princeton)

On Clustering Using Random Walks 18
David Harel, Yehuda Koren (Weizmann)

An Introduction to Decidability of DPDA Equivalence 42
Colin Stirling (Edinburgh)

Semidefinite Programming Based Approximation Algorithms 57
Uri Zwick (Tel Aviv)

Contributed Papers

Hard Sets and Pseudo-random Generators for Constant Depth Circuits ... 58
Manindra Agrawal (IIT Kanpur)

The First-Order Isomorphism Theorem 70
Manindra Agrawal (IIT Kanpur)

Thresholds and Optimal Binary Comparison Search Trees 83
Richard Anderson (Washington), Sampath Kannan (AT&T/Pennsylvania), Howard Karloff (AT&T/Georgia Tech), Richard E. Ladner (Washington)

Distributed LTL Model Checking Based on Negative Cycle Detection 96
Luboš Brim, Ivana Černá, Pavel Krčál, Radek Pelánek (Masaryk)

Computability and Complexity Results for a Spatial Assertion Language for Data Structures 108
Cristiano Calcagno (Queen Mary, DISI), Hongseok Yang (KAIST), Peter W. O'Hearn (Queen Mary)

Using Nondeterminism to Design Efficient Deterministic Algorithms 120
Jianer Chen, Donald K. Friesen (Texas A&M), Weijia Jia (Hong Kong), Iyad A. Kanj (DePaul)

Liveness Verification of Reversal-Bounded Multicounter Machines with a Free Counter 132
Zhe Dang (Washington State), Oscar H. Ibarra (Santa Barbara), Pierluigi San Pietro (Milano)

A Mechanically Verified Compiling Specification for a Lisp Compiler 144
Axel Dold, Vincent Vialard (Ulm)

Beyond Regular Model Checking 156
Dana Fisman, Amir Pnueli (Weizmann)

Relations Between Communication Complexity, Linear Arrangements, and Computational Complexity 171
Jürgen Forster (Bochum), Matthias Krause (Mannheim), Satyanarayana V. Lokam (Princeton), Rustam Mubarakzjanov (Trier), Niels Schmitt (Bochum), Hans Ulrich Simon (Bochum)

Optimal, Output-Sensitive Algorithms for Constructing Upper Envelope of Line Segments in Parallel 183
Neelima Gupta, Sumit Chopra (Delhi), Sandeep Sen (IIT Delhi)

List Decoding from Erasures: Bounds and Code Constructions 195
Venkatesan Guruswami (Berkeley)

Verification of a Leader Election Algorithm in Timed Asynchronous Systems 207
Neeraj Jaggi, K. Gopinath (IISc)

Efficient Addition on Field Programmable Gate Arrays 219
Andreas Jakoby (Lübeck), Christian Schindelhauer (Paderborn)

The Directed Minimum-Degree Spanning Tree Problem 232
Radha Krishnan, Balaji Raghavachari (Dallas)

I/O-Efficient Batched Range Counting and Its Applications to Proximity Problems 244
Tamás Lukovszki (Paderborn), Anil Maheshwari, Norbert Zeh (Carleton)

Beyond Message Sequence Graphs 256
P. Madhusudan (CMI, Chennay), B. Meenakshi (IMSc, Chennay)

Grouping Techniques for One Machine Scheduling Subject to Precedence Constraints 268
Monaldo Mastrolilli (IDSIA)

Properties of Distributed Timed-Arc Petri Nets 280
Mogens Nielsen (BRICS), Vladimiro Sassone (Sussex), Jiří Srba (BRICS)

From Falsification to Verification 292
Doron Peled (Austin), Amir Pnueli (Weizmann), Lenore Zuck (NYU)

On Polynomial Representations of Boolean Functions Related to Some Number Theoretic Problems 305
Erion Plaku (Clarkson), Igor E. Shparlinski (Macquarie)

Range Allocation for Equivalence Logic 317
Amir Pnueli (Weizmann), Yoav Rodeh, Ofer Shtrichman (Weizmann/IBM Haifa)

Rewrite Closure for Ground and Cancellative AC Theories 334
Ashish Tiwari (SRI)

Author Index **347**

When Worlds Collide: Derandomization, Lower Bounds, and Kolmogorov Complexity

Eric Allender

Department of Computer Science
Rutgers University
Piscataway, NJ 08854-8019, USA
allender@cs.rutgers.edu
http://www.cs.rutgers.edu/~allender

Abstract. This paper has the following goals:
- To survey some of the recent developments in the field of derandomization.
- To introduce a new notion of time-bounded Kolmogorov complexity (KT), and show that it provides a useful tool for understanding advances in derandomization, and for putting various results in context.
- To illustrate the usefulness of KT, by answering a question that has been posed in the literature, and
- To pose some promising directions for future research.

1 Introduction

If one were to have taken a poll among complexity theoreticians in 1985, asking if probabilistic algorithms are more powerful than deterministic ones (i.e., if P = BPP) I feel sure that a clear majority would have thought it more likely that BPP $\neq$ P. Similarly, during the late 1980's when the "Arthur-Merlin" classes MA and AM were introduced [1] it is fair to say that most people in the community thought it most likely that MA and AM were *not* merely new names for NP. A poll taken today would, I believe, show a majority believing BPP = P and MA = AM = NP.

Such swings of opinion are a sure sign of important progress in the field. (Consider the likely outcome of taking similar polls about whether NL=coNL, or whether IP=PSPACE, before and after the announcements that these problems had been settled [2,3,4,5].) In the case of BPP, MA, and AM, our new understanding of these classes can be credited to advances in the field of *derandomization* – a field that is so large and active that I will not pretend to be able to survey it here.

Rather, I will focus on a few recent and exciting developments in the field of derandomization, and I will show how these developments can be understood in the context of *time-bounded Kolmogorov complexity*. This has the additional

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 1–15, 2001.
© Springer-Verlag Berlin Heidelberg 2001

benefit of shedding light on the process of trying to prove lower bounds on the circuit complexity of various computational problems. In particular, I will highlight connections that exist between derandomization, Kolmogorov complexity, and the *natural proofs* framework of [6].

In this article, most proofs are sketched or omitted. A more detailed article describing this work and related results is in preparation with Detlef Ronneburger.

2 Ancient History

I assume that the reader is familiar with the complexity classes P, NP, MA, AM, and BPP. As has become customary, E and NE refer to DTIME($2^{O(n)}$) and NTIME($2^{O(n)}$), respectively, whereas EXP and NEXP refer to DTIME($2^{n^{O(1)}}$) and NTIME($2^{n^{O(1)}}$), respectively. Computations performed by circuits of AND, OR, and NOT gates having depth $O(1)$ and polynomial size define the class AC^0. When I refer to AC^0 I shall always refer to DLOGTIME-Uniform AC^0 (sometimes also called U_E-uniform AC^0) unless I explicitly mention non-uniform circuits. Occasionally, it is useful to refer to circuit size bounds other than polynomial. For instance, $\mathrm{AC}^0(2^{\epsilon n})$ refers to the class of languages accepted by circuit families of AND, OR, and NOT gates having depth $O(1)$ and size bounded by $2^{\epsilon n}$. P/poly is the class of languages for which there exist circuit families of polynomial size (with no restriction on circuit depth). For background on complexity classes and uniformity, consult standard texts such as [7,8,9].

A set L is *P-printable* if there is a function that, on input n, produces in time $n^{O(1)}$ a list of all elements of L having length at most n. P-printable sets were defined in [10] and were studied in [11] and elsewhere.

It remains an open question if NEXP is contained in P/poly. Perhaps more surprisingly, it remains unknown if there exists some $\epsilon < 1$ such that NEXP is contained in $\mathrm{AC}^0(2^{\epsilon n})$. (The lower bounds for the Parity and Majority functions [12] provide examples of problems in P that lie outside of non-uniform $\mathrm{AC}^0(2^{n^{\epsilon}})$, but in order to find examples of problems requiring larger circuits, currently we are forced to look outside of NEXP.)

The exponential-time analog of the P=NP problem is the question of whether or not EXP=NEXP. The reader probably knows that P=NP if and only if accepting computation paths of NP machines can be found in polynomial time. The situation at the level of exponential time is less clear, however. This motivates the following definition.

A *NEXP search problem* is a problem where, for a given nondeterministic Turing machine running in time 2^{n^c} for some c, one takes as input a string x and produces as output an accepting computation of M on input x (if such a string exists). (In the case where the running time is $2^{O(n)}$ we call this an *NE search problem.*) It is clear that if every NEXP search problem is solvable in deterministic exponential time, then NEXP = EXP; however the converse is not known to hold [13,14,15].

2.1 Levin's Kt Complexity

There can be no doubt that the question of how to find accepting computations of a nondeterministic machine is of central importance to computer science. Levin observed that there is an easy-to-compute ordering on Σ^* giving an essentially-optimal search strategy [16]. This can be stated more formally by means of Levin's formulation of *time-bounded Kolmogorov complexity*.

Definition 1 (Levin). *Let U be a universal Turing machine. Define* $\mathrm{Kt}(x)$ *to be* $\min\{|d| + \log t \ : \ U(d) = x \text{ in at most } t \text{ steps}\}$.

Note that $\log |x| \leq \mathrm{Kt}(x) \leq |x| + O(\log |x|)$.

The elements of Σ^* can be enumerated in order of increasing $\mathrm{Kt}(x)$, and Levin observed that this ordering yields essentially the fastest way to search for accepting computations. The following definitions (from [17,18]) will be useful in stating certain correspondences.

Definition 2. *Let $L \subseteq \Sigma^*$. Define* $\mathrm{Kt}_L(n)$ *to be* $\min\{\mathrm{Kt}(x) \ : \ x \in L^{=n}\}$.

(In [17,18] the function Kt_L was called K_L.) Here and elsewhere, $L^{=n}$ is the set of all strings in L of length n. If there is no string in L of length n, then $\mathrm{Kt}_L(n)$ is undefined. When considering the rate of growth of a function $\mathrm{Kt}_L(n)$, the undefined values are not taken into consideration. Observe that, for every language L, $\log n \leq \mathrm{Kt}_L(n) \leq n + O(\log n)$.

At this point, we can state some connections between the complexity of NEXP search problems, and time-bounded Kolmogorov complexity. The following statements are almost identical to observations in [17,18], and they derive from Levin's original insight about Kt-complexity.

Theorem 1. *The following are equivalent:*

- *All* NEXP *search problems are* EXP*-solvable.*
- *For all L in* P, $\mathrm{Kt}_L(n) = \log^{O(1)} n$.

Theorem 2. *The following are equivalent:*

- *For every $\epsilon > 0$, all* NE *search problems are solvable in time* $2^{2^{\epsilon n}}$.
- *For every $\epsilon > 0$ and for every L in* P, $\mathrm{Kt}_L(n) = O(n^\epsilon)$.

Since most complexity theoreticians conjecture that NEXP requires doubly exponential time, it follows that most complexity theoreticians conjecture that there are languages L in P such that $\mathrm{Kt}_L(n)$ grows nearly linearly.

On the other hand, most complexity theoreticians also conjecture that pseudorandom generators exist. As the following paragraphs explain, it follows that most complexity theoreticians need to conjecture that $\mathrm{Kt}_L(n)$ grows *slowly* if L is in P (or even in P/poly) and has very many strings in it.

For this paper, it will not be necessary to give precise definitions of pseudorandom generators. (The cited references provide details for the interested reader.) Recall only that a pseudorandom generator takes a short *seed* as input,

and produces a long *pseudorandom* output. In many applications we are interested in the case where pseudorandom output of length n is produced in time polynomial in n. In this case, note that the output of a pseudorandom generator is a string of small Kt-complexity.

A pseudorandom generator is said to be *secure* if, for any circuit C of size s, the probability that C accepts a random string of length n is within ϵ of the probability that C accepts a pseudorandom string of length n. (The parameters s and ϵ vary according to the security requirements.) Thus in particular note that if one has a secure pseudorandom generator and a circuit of polynomial size that accepts at least $2^n/n$ inputs of length n (i.e., the circuit accepts a "dense" subset of Σ^n), then it should also accept quite a few pseudorandom strings of length n. Hence it was observed in [17,18] that if secure pseudorandom generators exist, then for all dense languages L in P/poly, $\mathrm{Kt}_L(n) = O(n^\epsilon)$ or $\mathrm{Kt}_L(n) = O(\log n)$ (depending on the security assumptions that were made about the pseudorandom generators).

We have seen that there is reason to be interested in the rate of growth that is possible for functions Kt_L when L is in P. One might also ask about Kt_L when L is chosen from some other complexity class. The following observation (essentially identical to an observation credited to Russo in [11]) is of interest in this regard.

Theorem 3. *The following are equivalent:*

- *For every L in* P, $\mathrm{Kt}_L(n) = O(\log n)$.
- *For every L in* NP, $\mathrm{Kt}_L(n) = O(\log n)$.
- *For every L in* AC^0, $\mathrm{Kt}_L(n) = O(\log n)$.

Similarly (by simply considering the (string,witness) pairs for a language in NP) it is easy to show:

Theorem 4. *The following are equivalent:*

- *There is a language L in* P *and an $\epsilon > 0$ such that for all large n $\mathrm{Kt}_L(n)$ is defined and $\mathrm{Kt}_L(n) > n^\epsilon$.*
- *There is a language L in* NP *and an $\epsilon > 0$ such that for all large n $\mathrm{Kt}_L(n)$ is defined and $\mathrm{Kt}_L(n) > n^\epsilon$.*
- *There is a language L in* AC^0 *and an $\epsilon > 0$ such that for all large n $\mathrm{Kt}_L(n)$ is defined and $\mathrm{Kt}_L(n) > n^\epsilon$.*

3 A Brief Discussion of Derandomization

One of the best examples of derandomization is provided by the lovely work of Impagliazzo and Wigderson:

Theorem 5. *[19] If there is a language $A \in$ E and a constant $\epsilon > 0$ such that, for all large n, there is no circuit of size $2^{\epsilon n}$ accepting $A^{=n}$, then* BPP = P.

Alternate proofs of this theorem of [19] can be found in [20].

The hypothesis to Theorem 5 seems very likely to be true. Indeed, it seems possible that there are problems in smaller complexity classes than E (including perhaps SAT itself) that require exponential-size circuits. It also seems likely that E contains problems that are significantly harder than this hypothesis assumes. For instance, it seems likely that E requires circuits of this size, even if the circuits are allowed to have "oracle gates" for SAT [21]. It is shown in [22] that this stronger hypothesis about the complexity of problems in E implies that AM = MA = NP. In [22] the same conclusion AM=NP is shown to follow from a weaker assumption: namely that the "hard" set A is in NE∩coNE. This hypothesis was subsequently weakened further by [23].

If one is willing to settle for a weaker conclusion than BPP=P, then it suffices to start with a much weaker assumption.

Definition 3. *Let t be a time bound. Define* io-DTime($t(n)$) *to be the class of languages L for which there exists a language $A \in$ DTime($t(n)$) with the property that, for infinitely many n, $L^{=n} = A^{=n}$.*

Theorem 6. *[24] If* EXP *is not in* P/poly*, then*

$$\text{BPP} \subseteq \bigcap_{\epsilon} \text{io-DTime}(2^{n^\epsilon}).$$

This is an example of "limited derandomization"; while it does not yield a polynomial-time algorithm for BPP problems, it does improve over the trivial exponential-time upper bound (at least for infinitely many input lengths). It is difficult to imagine that languages in BPP could be easy to recognize on some input lengths and difficult to recognize on other input lengths. Nonetheless, it is still not known if "io-Dtime" can be replaced by "Dtime" in the preceding theorem.

The conclusion of Theorem 6 implies that EXP is not contained in BPP – but it is not known if the ability to do limited derandomization implies that EXP is not in P/poly. That is, lower bounds are *sufficient* for limited derandomization, but lower bounds are not known to be *necessary* for derandomization. Stated another way, it would be interesting to know if the nonexistence of circuit lower bounds for problems in EXP (i.e., EXP $\subseteq$ P/poly) implies that no derandomization is possible (i.e., that BPP = EXP).

In contrast, it is shown by Impagliazzo, Kabanets, and Wigderson (in part credited by them to van Melkebeek) that lower bounds for NEXP are *necessary and sufficient* for limited derandomization of MA:

Theorem 7. *[25] The following are equivalent:*

- NEXP $\not\subseteq$ P/poly
- NEXP $\neq$ MA
- MA $\subseteq \bigcap_\epsilon$ io(NTime(2^{n^ϵ})/n^ϵ).

Central to the approach taken in [25] is the strategy of searching for solutions to NEXP-search problems by looking for "easy witnesses" first, where a string of length 2^n is considered to be an "easy witness" if it is the truth table of a function with small circuit complexity.

This leads us to the following two questions:

Question 1: We already know that Kt complexity defines an optimal search strategy for NEXP search problems. How does this compare to the "easy witness" strategy of [25]?

Question 2: Is the notion of "easy witnesses" also related to a variant of Kolmogorov complexity?

4 A New Definition of Time-Bounded Kolmogorov Complexity

To try to find an answer to these questions, we are led to a new definition of time-bounded Kolmogorov complexity. The definition below is a slight variant on a definition introduced by Antunes, Fortnow, and van Melkebeek in [26].

Definition 4. *Let U be a universal Turing machine. Define* $\mathrm{KT}(x)$ *to be*

$$\min\{|d| + t : \text{for all } i \leq |x|, U(d,i) = x_i \text{ in at most } t \text{ steps}\}.$$

(*A note on notation:* The capital "T" in KT calls attention to the fact that the time bound has much more influence than in Kt complexity.) In the preceding definition, note that the string d is "almost" a description of of x in the usual sense; it is possible that U cannot determine the length of x from d. Of course, by adding $O(\log n)$ to the length of d, we could require that $U(d,i) = \perp$ for all $i > |x|$ (thus obtaining a description of x in the usual sense). Logarithmic terms are insignificant for our applications, and thus we use the simpler definition.

Observe that the KT and Kt measures can be generalized to obtain KT^A and Kt^A for oracles A, by giving U access to an oracle. It turns out that this is useful in clarifying the relationship between Kt and KT complexity.

Theorem 8 (Ronneburger). *[27] Let A be complete for* E *under linear-time reductions. Then there is a $k \in \mathbb{N}$ such that* $\mathrm{Kt}(x)/k < \mathrm{KT}^A(x) < k\mathrm{Kt}(x)$. *In other words,* $\mathrm{Kt} = \Theta(\mathrm{KT}^A)$.

At first glance, the definition of KT complexity seems to have quite a different flavor than Levin's Kt complexity, especially because Levin's definition uses the log of the running time, and KT complexity uses running time. However, the preceding theorem shows that one can view Kt complexity as merely a variation of KT complexity; Kt complexity is KT complexity relative to E.

Next, let us observe that KT complexity captures the essential character of the "easy witness" approach of [25].

Theorem 9. *Let x be a string of length 2^n (which we can view as the truth table of a function f on inputs of length n), and let A be any oracle. If* $\mathrm{KT}^A(x) < m$ *then there is an A-circuit of size $O(m^3)$ computing f. Conversely, if there is an A-circuit of size m computing f, then* $\mathrm{KT}^A(x) = O(m \log m)$.

That is, $\mathrm{KT}(x)$ is an approximation to the circuit size required to compute the function given by the truth-table x.

It will be useful for us to define an analog to the function Kt_L.

Definition 5. *Let $L \subseteq \Sigma^*$. Define $\mathrm{KT}_L(n)$ to be $\min\{\mathrm{KT}(x) \ : \ x \in L^{=n}\}$.*

The following two theorems have the same easy proofs as the corresponding observations for Kt_L.

Theorem 10. *The following are equivalent:*

- *For every L in* P, $\mathrm{KT}_L(n) = \log^{O(1)} n$.
- *For every L in* NP, $\mathrm{KT}_L(n) = \log^{O(1)} n$.
- *For every L in* AC^0, $\mathrm{KT}_L(n) = \log^{O(1)} n$.

Theorem 11. *The following are equivalent:*

- *There is a language L in* P *and an $\epsilon > 0$ such that for all large n $\mathrm{KT}_L(n)$ is defined and $\mathrm{KT}_L(n) > n^\epsilon$.*
- *There is a language L in* NP *and an $\epsilon > 0$ such that for all large n $\mathrm{KT}_L(n)$ is defined and $\mathrm{KT}_L(n) > n^\epsilon$.*
- *There is a language L in* AC^0 *and an $\epsilon > 0$ such that for all large n $\mathrm{KT}_L(n)$ is defined and $\mathrm{KT}_L(n) > n^\epsilon$.*

4.1 How Do KT and Kt Compare?

Proposition 1. $\frac{1}{3}\mathrm{Kt}(x) \leq \mathrm{KT}(x) \leq 2^{1+\mathrm{Kt}(x)}$.

It is reasonable to conjecture that one of the two inequalities above is essentially tight. This leads us to formulate the two hypotheses below.

Hypothesis A: $\mathrm{KT}(x) = \mathrm{Kt}(x)^{O(1)}$.

Hypothesis B: There is some $\epsilon > 0$ such that for all large n there is a string $x \in \Sigma^n$ such that $\mathrm{KT}(x) > 2^{\epsilon \mathrm{Kt}(x)}$.

Hypothesis A says that the first inequality is essentially tight, and Hypothesis B says that the second inequality is essentially tight.

In the following section, we will see that each of these hypotheses has already been the object of a great deal of attention in the complexity theory community.

5 The Hypotheses Are Familiar

First, let us consider Hypothesis A, which states that there is not much difference between KT and Kt. This is equivalent to a number of other statements, all of which seem highly unlikely.

Theorem 12. *The following are equivalent.*

1. *Hypothesis A*
2. $\mathrm{EXP} \subseteq \mathrm{P/poly}$.

3. *For all* P*-printable sets* L, $\mathrm{KT}_L(n) = \log^{O(1)} n$.
4. *There exists a dense set* $L \in \mathrm{P/poly}$ *and some* $\epsilon > 0$ *such that for all large* n, $\mathrm{Kt}_L(n)$ *is defined and* $\mathrm{Kt}_L(n) > n^\epsilon$.

Proof. The equivalence (*1.* $\iff$ *2.*) is straightforward. (Merely examine the KT complexity of prefices of the characteristic sequence of a complete language for E.) Essentially the same observations suffice to show (*3.* $\iff$ *1.*)

The proof of Theorem 6 (see [24]) actually establishes the implication (*4.* $\implies$ *2.*) That is, the authors of [24] show that if EXP $\not\subseteq$ P/poly, then the Nisan-Wigderson pseudorandom generator ([28]) can be used with seed length n^ϵ to simulate any BPP algorithm. In fact, their analysis shows that the generator must output a string accepted by any small circuit that accepts many strings, which is precisely the negation of condition *4.*

For the implication (*2.* $\implies$ *4.*) we use an argument of [29]. Assume that condition *4.* fails to hold. That is, for every dense $L \in \mathrm{P/poly}$ and every $\epsilon > 0$, there are infinitely many $x \in L$ such that $\mathrm{Kt}(x) < |x|^\epsilon$. This yields a "hitting set generator;" i.e., a function G_ϵ running in time $2^{O(n^\epsilon)}$ that, on input n, outputs a list of strings such that any dense language in P/poly must accept one of the strings in the list. The complement of the range of of G_ϵ is dense, and is in E. If it were in P/poly it would contradict the existence of G_ϵ.

The equivalence (*3.* $\iff$ *4.*) is intriguing. It says that assuming that one class of sets has *small* Kolmogorov complexity is equivalent to another class of sets having *large* Kolmogorov complexity (using slightly different notions of time-bounded Kolmogorov complexity). This tension between high and low complexity is responsible for many of the most exciting aspects of derandomization.

Klivans and van Melkebeek show that the condition EXP $\not\subseteq$ P/poly is sufficient to carry out limited derandomization of MA. If a stronger hypothesis is used, (i.e., one assumes that some P-printable set has large $\mathrm{KT}^{\mathrm{SAT}}$ complexity instead of merely large KT complexity) then one obtains a derandomization of AM [22].

Let us now turn our attention from Hypothesis A to Hypothesis B.

Theorem 13. *The following are equivalent:*

1. *Hypothesis B*
2. *There exists a* P*-printable set* L *and an* $\epsilon > 0$ *such that for all large* n, $\mathrm{KT}_L(n)$ *is defined and* $\mathrm{KT}_L(n) > n^\epsilon$.
3. *For all dense* L *in* $\mathrm{P/poly}, \mathrm{Kt}_L(n) = O(\log n)$.
4. *There is a language* $A \in \mathrm{E}$ *and a constant* $\epsilon > 0$ *such that, for all large* n, *there is no circuit of size* $2^{\epsilon n}$ *accepting* $A^{=n}$.
5. *Efficient pseudorandom generators* G *exist, such that* $G : \Sigma^{k \log n} \to \Sigma^n$.
6. *Efficient hitting set generators exist.*

That is, in particular, Hypothesis B is equivalent to the hypothesis of Theorem 5.

Proof. Items *4.* through *6.* in the list above have been observed before to be equivalent [30,31,29]. The equivalence (*1.* $\iff$ *2.* $\iff$ *4.*) is straightforward and similar to the proof of Theorem 12. It remains only to consider condition *3.*

Condition *3.* is easily seen to be equivalent to the existence of a hitting set generator (i.e., a polynomial-time-computable function that, on input n, outputs a list S of strings of length n with the property that any circuit of size n accepting at least $2^n/n$ elements of Σ^n must accept an element of S). If we assume *3.* is true, then the routine that lists all strings of length n having Kt complexity $O(\log n)$ is a hitting set generator. Conversely, if we assume that hitting set generators exist, note that all strings produced by the generator have Kt complexity $O(\log n)$.

The preceding theorems gave equivalent ways to view the question of whether or not P-printable sets can have large KT-complexity. The following theorem shows that it is of interest to consider the KT-complexity of arbitrary sets in P.

Theorem 14. *The following are equivalent:*

1. *For all* $L \in \mathrm{P}, \mathrm{KT}_L(n) = \log^{O(1)} n$.
2. *All* NEXP *search problems are solvable in* P/poly.
3. NEXP $\subseteq$ P/poly
4. NEXP = MA
5. MA $\not\subseteq \bigcap_\epsilon \mathrm{io}(\mathrm{NTime}(2^{n^\epsilon})/n^\epsilon)$.

Proof. The equivalence (*1.* $\iff$ *2.*) is straightforward. Equivalences (*3.* $\iff$ *4.* $\iff$ *5.*) and implication (*3.* $\implies$ *2.*) were established in [25]. The remaining implication (*2.* $\implies$ *3.*) is trivial.

6 The Question of Density

It is obvious that there are sets L in P/poly (and even in non-uniform AC^0) with high Kt_L complexity (and hence with high KT_L complexity), since for each length n one can select a string x_n of length n with high Kt-complexity, and build a circuit that accepts exactly x_n. However, as observed earlier, dense languages L in P/poly must have low Kt_L complexity, or else secure pseudorandom generators do not exist.

Theorems 3, 4, 10, and 11 show that the Kolmogorov complexity of sets in P and AC^0 are closely related. It is natural to wonder if the Kolmogorov complexity of *dense* sets in P/poly and non-uniform AC^0 are also related. No such relationship is known, although dense sets in AC^0 provide one of the few examples where we are able to say something nontrivial about Kt and KT complexity.

It was essentially observed by [32] that the derandomization results of Nisan and Wigderson [28] can be used to show that, for dense L in AC^0, $\mathrm{Kt}_L(n) = \log^{O(1)} n$. Closer examination of the Nisan-Wigderson generator shows that an even stronger conclusion holds:

Theorem 15. *Let* L *be a dense language in* AC^0. *Then* $\mathrm{KT}_L(n) = \log^{O(1)} n$.

For even slightly larger classes (such as $\mathrm{AC}^0[2]$) no nontrivial bound on the KT_L or Kt_L complexity is known.

It is natural to wonder if the bound of $\log^{O(1)} n$ in the preceding theorem can be improved to $O(\log n)$. The results of Agrawal [33] can be used to show that this question is equivalent to a question in circuit complexity:

Theorem 16. *The following are equivalent.*

- *For all dense L in non-uniform* AC^0*,* $\mathrm{Kt}_L(n) = O(\log n)$.
- *For some* $\epsilon > 0, \mathrm{E} \not\subseteq \mathrm{AC}^0(2^{\epsilon n})$.

7 Natural Proofs, Lower Bounds, and Kolmogorov Complexity

We have seen that there are good reasons to believe that dense sets in P/poly must have low Kt complexity, but we have not discussed any consequences of the existence of dense sets in P/poly with large KT complexity.

It turns out that this is exactly the question that was raised by Razborov and Rudich when they developed their framework of "natural proofs" to explain why certain approaches to proving circuit lower bounds appear to be doomed to failure [6]. Let us recall some of the definitions of Razborov and Rudich.

Definition 6. *[6] A* Combinatorial Property *of Boolean functions is a subset of* $\{\Sigma^{2^n} : n \in \mathbb{N}\}$*; that is, it is a set of truth tables.*

A combinatorial property C is useful against P/poly *if any language L in* P/poly *has the property that $\{n$: the truth table for $L^{=n}$ lies in $C\}$ is finite.*

Razborov and Rudich also consider a "largeness" condition; we shall discuss that later.

Note that, by Theorem 9, if $\mathrm{KT}(x)$ is large, then there is some $i < |x|$ such that $x0^i$ is the truth table of a function requiring large circuits. From this simple observation, we can see that a language L where $\forall k \forall^\infty n \mathrm{KT}_L(n) \geq \log^k n$ is the same thing as a combinatorial property useful against P/poly.

Impagliazzo, Kabanets, and Wigderson observed in [25] that if there is a combinatorial property in NP that is useful against P/poly, then $\mathrm{NEXP} \not\subseteq \mathrm{P/poly}$. They then asked if the existence of a combinatorial property in P that is useful against P/poly implies that $\mathrm{EXP} \not\subseteq \mathrm{P/poly}$. However, as in the proof of Theorem 10, we can see that there is a combinatorial property in NP that is useful against P/poly if and only if such a property exists in AC^0.

The other crucial ingredient that a combinatorial property needs in order to be "natural" (in the framework of Razborov and Rudich) is that it needs to be dense. (That is, it must contain at least $2^N/N^k$ strings, for all N of the form 2^n. All theorems in this paper involving "dense" sets carry over if we change our density requirement from $2^n/n$ to $2^n/n^{O(1)}$.)

Razborov and Rudich showed that, under strong (but widely-believed) assumptions about the existence of secure pseudorandom generators, there is no

dense combinatorial property in P/poly that is useful against P/poly. Hence, under the hypothesis of [6], for all dense $L \in \mathrm{P/poly}, \mathrm{KT}_L(n) = \log^{O(1)} n$.

Note that there are very close connections between the notion of Natural Proofs and certain aspects of resource-bounded measure theory [34,35]. There is not space here to elaborate on these connections. However, this suggests that there may be interesting implications between time-bounded Kolmogorov complexity and resource-bounded measure that deserve to be explored.

8 Random Strings

The canonical dense sets with large Kolmogorov complexity are the sets containing *all* of the random strings. For our purposes, it will suffice to consider the following two sets:

- R_{KT} is defined to be $\{x|\mathrm{KT}(x) \geq |x|/2\}$.
- R_{Kt} is defined to be $\{x|\mathrm{Kt}(x) \geq |x|/2\}$.

Similar (but not identical) notions of time-bounded randomness have been considered before [36,37,38,39].

R_{KT} is in coNP, and R_{Kt} is in E. It seems natural to conjecture that these upper bounds are essentially optimal. However, there are significant obstacles to showing that there are no smaller complexity classes containing these languages. Most significantly, there is reason to believe that these languages are *not complete* for coNP and E, respectively. For some notions of completeness, it is even possible to prove that this is the case.

Theorem 17. *R_{Kt} is not hard for* E *under polynomial-time many-one reductions.*

Proof. We provide only a sketch. Let T be a subset of 0^* that is in E but not in P. Let f be any polynomial-time function. Note that $\mathrm{Kt}(f(0^n)) = O(\log n)$. Thus $f(0^n)$ is not in R_{Kt} unless $f(0^n)$ is *very* short – in which case membership in R_{Kt} can be determined in polynomial time.

Essentially the same proof also shows that R_{Kt} is not complete under polynomial-time truth-table reductions.

Buhrman and Mayordomo show that a related (but not identical) language is not hard for E even under polynomial-time Turing reductions (and in fact they obtain even stronger conclusions relating to resource-bounded measure) [40]. However, their argument does not seem to extend to R_{Kt}.

By Theorem 15, neither R_{Kt} nor R_{KT} is in (non-uniform) AC^0. Somewhat surprisingly, as the deadline for submission of this paper approaches I am unable to find or construct any other lower bounds on the complexity either of these problems (other than conditional lower bounds, such as the argument in [6] showing that these sets are not in P/poly if strong pseudorandom generators exist).

Ko presents oracles relative to which a set closely related to R_{KT} is not complete for coNP. The recent proof by Agrawal, showing that all sets complete

for NP under AC^0 many-one reductions are AC^0-isomorphic [41] can be used to show that R_{Kt} is not hard even for some very small subclasses of P. (In addition, it provides conditions where R_{KT} also fails to be hard).

Theorem 18. *R_{Kt} is not hard for TC^0 under AC^0-many-one reductions. Also, if SAT has circuits of size $2^{n^{o(1)}}$, then R_{KT} is not hard for TC^0 under AC^0-many-one reductions.*

Proof. Again, we provide only a sketch. Agrawal shows in [41] that any set hard for TC^0 under AC^0 reductions is hard under length-increasing NC^0 reductions where the connections in the NC^0 circuits are computable in AC^0. Thus, in particular, the output of the reduction has small Kt complexity and cannot lie in R_{Kt}. For KT complexity, we require an additional assumption. Under the assumption that SAT has small circuits, there is a small circuit that takes i as input, and produces an NC^0 circuit for the i-th bit of the reduction as output. Now one can show that the output of the reduction must have small KT complexity, and hence foil the reduction.

For instance, if P=NP, then $R_{Kt} \notin P$ (by Theorem 13) and $R_{KT} \in P$, but R_{KT} is not hard for TC^0 under AC^0 many-one reductions.

Cai and Kabanets study a question of whether a set closely related to R_{KT} is in P or is NP-complete. For instance, if R_{KT} is complete under length-increasing polynomial-time reductions, then there is a P-printable set L such that Kt_L grows very quickly, and hence one obtains all of the consequences listed in Theorem 13.

9 Conclusions

Kolmogorov complexity has been very useful in attacking a wide variety of problems in computer science and mathematics. It has not found much application (yet) in the field of derandomization. (A notable exception is the work of Trevisan [42].) It is hoped that the definitions provided here will be useful in formulating new lines of attack on these problems.

In particular, it seems quite likely to me that better bounds can be proved on the complexity of R_{KT}. It should be possible to prove sublinear bounds on the growth of Kt_L for dense sets L in $AC^0[2]$. I feel sure that there are new and interesting implications waiting to be discovered involving the KT_L and Kt_L complexity of various classes of sets. Since these notions have tight connections to the theory of natural proofs (and hence to resource-bounded measure), I think it is possible that investigation of these questions will have application not only to the study of derandomization, but throughout complexity theory.

Acknowledgments

This work was supported by National Science Foundation grant number CCR-0104823. I thank Russell Impagliazzo, Lance Fortnow, Detlef Ronneburger, Va-

lentine Kabanets, and Dieter van Melkebeek for helpful comments and for pointing out errors in an earlier draft of this paper. I also thank Detlef Ronneburger for allowing me to report on our ongoing work here.

References

1. Babai, L., Moran, S.: Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. Journal of Computer and System Sciences **36** (1988) 254–276
2. Immerman, N.: Nondeterministic space is closed under complement. SIAM Journal on Computing **17** (1988) 935–938
3. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Informatica **26** (1988) 279–284
4. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM **39** (1992) 859–868
5. Shamir, A.: IP = PSPACE. Journal of the ACM **39** (1992) 869–877
6. Razborov, A.A., Rudich, S.: Natural proofs. Journal of Computer and System Sciences **55** (1997) 24–35
7. Papadimitriou, C.: Computational Complexity. Addison-Wesley, New York (1994)
8. Du, D.Z., Ko, K.I.: Theory of Computational Complexity. Wiley-Interscience, New York (2000)
9. Vollmer, H.: Introduction to Circuit Complexity. Springer-Verlag (1999)
10. Hartmanis, J., Yesha, Y.: Computation times of NP sets of different densities. Theoretical Computer Science **34** (1984) 17–32
11. Allender, E., Rubinstein, R.: P-printable sets. SIAM J. Comput. **17** (1988) 1193–1202
12. Håstad, J.: Computational Limitations of Small Depth Circuits. MIT Press, Cambridge (1988)
13. Allender, E., Watanabe, O.: Kolmogorov complexity and degrees of tally sets. Information and Computation **86** (1990) 160–178
14. Impagliazzo, R., Tardos, G.: Decision versus search problems in super-polynomial time. In: IEEE Symposium on Foundations of Computer Science (FOCS). (1989) 222–227
15. Buhrman, H., Fortnow, L., Laplante, S.: Resource-bounded Kolmogorov complexity revisited. SIAM Journal on Computing (2001) to appear; a preliminary version appeared in STACS 1997.
16. Levin, L.A.: Randomness conservation inequalities; information and independence in mathematical theories. Information and Control **61** (1984) 15–37
17. Allender, E.: Some consequences of the existence of pseudorandom generators. Journal of Computer and System Sciences **39** (1989) 101–124
18. Allender, E.: Applications of time-bounded kolmogorov complexity in complexity theory. In: Osamu Watanabe (Ed.), Kolmogorov Complexity and Computational Complexity. Springer-Verlag (1992) 4–22
19. Impagliazzo, R., Wigderson, A.: $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In: ACM Symposium on Theory of Computing (STOC). (1997) 220–229
20. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the XOR lemma. Journal of Computer and System Sciences **62** (2001) 236–266

21. Arvind, V., Köbler, J.: On pseudorandomness and resource-bounded measure. Theoretical Computer Science **255** (2001) 205–221
22. Klivans, A., van Melkebeek, D.: Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In: ACM Symposium on Theory of Computing (STOC). (1999) 659–667
23. Miltersen, P.B., Vinodchandran, N.V.: Derandomizing Arthur-Merlin games using hitting sets. In: IEEE Symposium on Foundations of Computer Science (FOCS). (1999) 71–80
24. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP has subexponential time simulations unless EXPTIME has publishable proofs. Computational Complexity **3** (1993) 307–318
25. Impagliazzo, R., Kabanets, V., Wigderson, A.: In search of an easy witness: Exponential time vs. probabilistic polynomial time. In: IEEE Conference on Computational Complexity. (2001) 2–12
26. Antunes, L., Fortnow, L., van Melkebeek, D.: Computational depth. In: IEEE Conference on Computational Complexity. (2001) 266–273
27. Ronneburger, D.: Personal communication. (2001)
28. Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences **49** (1994) 149–167
29. Impagliazzo, R., Shaltiel, R., Wigderson, A.: Near-optimal conversion of hardness into pseudo-randomness. In: IEEE Symposium on Foundations of Computer Science (FOCS). (1999) 181–190
30. Fortnow, L.: Comparing notions of full derandomization. In: IEEE Conference on Computational Complexity. (2001) 28–34
31. Kabanets, V., Rackoff, C., Cook, S.: Efficiently approximable real-valued functions. In: Electronic Colloquium on Computational Complexity (ECCC) Report TR00-034. (2000) (www.eccc.uni-trier.de/eccc).
32. Zimand, M.: Large sets in AC^0 have many strings with low Kolmogorov complexity. Information Processing Letters **62** (1997) 165–170
33. Agrawal, M.: Hard sets and pseudo-random generators for constant depth circuits. In: Proceedings 21st Foundations of Software Technology and Theoretical Computer Science Conference (FST&TCS). Lecture Notes in Computer Science, Springer (2001) (these proceedings).
34. Regan, K.W., Sivakumar, D., Cai, J.Y.: Pseudorandom generators, measure theory, and natural proofs. In: IEEE Symposium on Foundations of Computer Science (FOCS). (1995) 26–35
35. Lutz, J.H.: Almost everywhere high nonuniform complexity. Journal of Computer and System Sciences **44** (1992) 220–258
36. Buhrman, H., Torenvliet, L.: Randomness is hard. SIAM Journal on Computing **30** (2001) 1485–1501
37. Ko, K.I.: On the complexity of learning minimum time-bounded Turing machines. SIAM Journal on Computing **20** (1991) 962–986
38. Hartmanis, J.: Generalized Kolmogorov complexity and the structure of feasible computations (preliminary report). In: IEEE Symposium on Foundations of Computer Science (FOCS). (1983) 439–445
39. Kabanets, V., Cai, J.Y.: Circuit minimization problem. In: ACM Symposium on Theory of Computing (STOC). (2000) 73–79
40. Buhrman, H., Mayordomo, E.: An excursion to the Kolmogorov random strings. Journal of Computer and System Sciences **54** (1997) 393–399

41. Agrawal, M.: The first-order isomorphism theorem. In: Proceedings 21st Foundations of Software Technology and Theoretical Computer Science Conference (FST&TCS). Lecture Notes in Computer Science, Springer (2001) (these proceedings).
42. Trevisan, L.: Construction of extractors using pseudo-random generators. Journal of the ACM (2001) to appear; a preliminary version appeared in STOC 1999.

Approximation Schemes for Geometric NP-Hard Problems: A Survey

Sanjeev Arora

Department of Computer Science
Princeton University
Princeton, NJ, USA
arora@cs.princeton.edu

Geometric optimization problems arise in many disciplines and are often NP-hard. One example is the famous *Traveling Salesman Problem* (TSP): given n points in the plane (more generally, in $\Re^d$), find the shortest closed path that visits them all.

Computing the optimum solution is NP-hard, which means that it has no efficient (i.e., polynomial-time) algorithm if P $\neq$ NP. Hence focus has shifted to designing *approximation algorithms*: algorithms that compute a solution whose cost is within a small multiplicative factor of the optimum. For instance, the classical Christofides heuristic can compute, for every TSP instance, a solution whose cost is within 50% of the cost of the optimum solution.

Work in the past few years has led to dramatically better approximation algorithms for the TSP and many other geometric problems such as *Steiner Tree*, *k-median, k-MST, facility location, minimum latency,* etc. All these problems are now known to have *approximation schemes*, which are algorithms that, for every $\epsilon > 0$, can compute a solution of cost at most $(1 + \epsilon)$ times the optimum.

The talk will survey these approximations schemes, and explain the ideas behind their design. The talk will be self-contained.

Bern and Eppstein [6] surveyed this field in their 1995 article. The author is preparing an up-to-date survey which should be available soon from his webpage at `http://www.cs.princeton.edu`. *The article will also list the remaining major open problems in the field.*

References

1. S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *JACM* **45**(5) 753-782, 1998. Preliminary versions in *Proceedings of 37th IEEE Symp. on Foundations of Computer Science*, 1996, and *Proceedings of 38th IEEE Symp. on Foundations of Computer Science*, 1997.
2. S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. *Proc. ACM Symposium on Theory of Computing*, 1999.
3. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for the Euclidean k-medians and related problems. In In *Proc. 30th ACM Symposium on Theory of Computing*, pp 106–113, 1998.
4. T. Asano, N. Katoh, H. Tamaki and T. Tokuyama. Covering Points in the Plane by k-Tours: Towards a Polynomial Time Approximation Scheme for General k. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pp 275–283, 1997.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 16–17, 2001.
© Springer-Verlag Berlin Heidelberg 2001

5. A. Barvinok, S. Fekete, D. S. Johnson, A. Tamir, G. J. Woeginger, R. Woodroofe. The maximum traveling salesman problem. *Merger of two papers in Proc. IPCO 1998 and Proc. ACM-SIAM SODA 1998.*
6. M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In [10].
7. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In J.F. Traub, editor, *Symposium on new directions and recent results in algorithms and complexity*, page 441. Academic Press, NY, 1976.
8. A. Czumaj and A. Lingas. A polynomial time approximation scheme for Euclidean minimum cost k-connectivity. *Proc. 25th Annual International Colloquium on Automata, Languages and Programming,* LNCS, Springer Verlag 1998.
9. W. Fernandez de la Vega and C. Kenyon. A randomized approximation scheme for metric MAX-CUT. *Proc. 39th IEEE Symp. on Foundations of Computer Science*, pp 468–471, 1998.
10. D. Hochbaum, ed. Approximation Algorithms for NP-hard problems. PWS Publishing, Boston, 1996.
11. S. G. Kolliopoulos and S. Rao. A nearly linear time approximation scheme for the Euclidean k-median problem. *LNCS*, vol.1643, pp 378–387, 1999.
12. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. The traveling salesman problem. John Wiley, 1985
13. J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple PTAS for geometric k-MST, TSP, and related problems. *SIAM J. Comp.*, **28**, 1999. Preliminary manuscript, April 30, 1996. To appear in *SIAM J. Computing.*
14. S. Rao and W. Smith. Approximating geometric graphs via "spanners" and "banyans." In *Proc. 30th ACM Symposium on Theory of Computing*, pp. 540–550, 1998.
15. A.Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner Problem. *Algorithmica*, **9:**463-470, 1993.

On Clustering Using Random Walks

David Harel and Yehuda Koren

Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
{harel,yehuda}@wisdom.weizmann.ac.il

Abstract. We propose a novel approach to clustering, based on deterministic analysis of random walks on the weighted graph associated with the clustering problem. The method is centered around what we shall call *separating operators*, which are applied repeatedly to sharpen the distinction between the weights of inter-cluster edges (the so-called separators), and those of intra-cluster edges. These operators can be used as a stand-alone for some problems, but become particularly powerful when embedded in a classical multi-scale framework and/or enhanced by other known techniques, such as agglomerative clustering. The resulting algorithms are simple, fast and general, and appear to have many useful applications.

1 Introduction

Clustering is a classical problem, applicable to a wide variety of areas. It calls for discovering natural groups in data sets, and identifying abstract structures that might reside there. Clustering methods have been used in computer vision [11,2], VLSI design [4], data mining [3], web page clustering , and gene expression analysis.

Prior literature on the clustering problem is huge, see e.g., [7]. However, to a large extent the problem remains elusive, and there is still a dire need for a clustering method that is natural and robust, yet very efficient in dealing with large data sets.

In this paper, we present a new set of clustering algorithms, based on deterministic exploration of random walks on the weighted graph associated with the data to be clustered. We use the similarity matrix of the data set, so no explicit representation of the coordinates of the data-points is needed. The heart of the method is in what we shall be calling *separating operators*, which are applied to the graph iteratively. Their effect is to 'sharpen' the distinction between the weights of inter-cluster edges (those that ought to separate clusters) and intra-cluster edges (those that ought to remain inside a single cluster), by decreasing the former and increasing the latter. The operators can be used on their own for some kinds of problems, but their power becomes more apparent when embedded in a classical multi-scale framework and when enhanced by other known techniques, such as agglomerative or hierarchical clustering.

The resulting algorithms are simple, fast and general. As to the quality of the clustering, we exhibit encouraging results of applying these algorithms to several

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 18–41, 2001.
© Springer-Verlag Berlin Heidelberg 2001

recently published data sets. However, in order to be able to better assess its usefulness, we are in the process of experimenting in other areas of application too.

2 Basic Notions

We use standard graph-theoretic notions. Specifically, let $G(V, E, w)$ be a weighted graph, which should be viewed as modeling a relation E over a set V of entities. Assume, without loss of generality, that the set of nodes V is $\{1, \ldots, n\}$. The w is a weighting function $w : E \longrightarrow \mathbb{R}^+$, that measures the similarity between pairs of items (a higher value means more similar). Let $S \subseteq V$. The set of nodes that are connected to some node of S by a path with at most k edges is denoted by $V^k(S)$. The degree of G, denoted by $deg(G)$, is the maximal number of edges incident to some single node of G. The subgraph of G induced by S is denoted by $G(S)$. The edge between i and j is denoted by $\langle i, j \rangle$. Sometimes, when the context is clear, we will write simply $\langle i, j \rangle$ instead of $\langle i, j \rangle \in E$.

A *random walk* is a natural stochastic process on graphs. Given a graph and a start node, we select a neighbor of the node at random, and 'go there', after which we continue the random walk from the newly chosen node. The probability of a transition from node i to node j, is

$$p_{ij} = \frac{w(i,j)}{d_i}$$

where $d_i = \sum_{\langle i,k \rangle} w(i,k)$ is the *weighted degree* of node i.

Given a weighted graph $G(V, E, w)$, the associated *transition matrix*, denoted by M^G, is the $n \times n$ matrix in which, if i and j are connected, the (i, j)'th entry is simply p_{ij}. Hence, we have

$$M^G_{ij} = \begin{cases} p_{ij} & \langle i, j \rangle \in E \\ 0 & \text{otherwise} \end{cases}$$

Now, denote by $P^k_{visit}(i) \in \mathbb{R}^n$ the vector whose j-th component is the probability that a random walk originating at i will visit node j in its k-th step. Thus, $P^k_{visit}(i)$ is the i-th row in the matrix $(M^G)^k$, the k'th power of M^G.

The *stationary distribution* of G is a vector $p \in \mathbb{R}^n$ such that $p \cdot M^G = p$. An important property of the stationary distribution is that if G is non-bipartite, then $P^k_{visit}(i)$ tends to the stationary distribution as k goes to ∞, regardless of the choice of i.

The *escape probability* from a source node s to a target node t, denoted by $P_{escape}(s, t)$, is defined as the probability that a random walk originating at s will reach t before returning to s. This probability can be computed as follows. For every $i \in V$, define a variable ρ_i satisfying:

$$\rho_s = 0, \qquad \rho_t = 1, \quad \text{and}$$
$$\rho_i = \sum_{\langle i,j \rangle} p_{ij} \cdot \rho_j \qquad \text{for } i \neq s,\ i \neq t$$

The values of ρ_i are calculated by solving these equations [1], and then the desired escape probability is given by:

$$P_{escape}(s,t) = \sum_{\langle s,i \rangle} p_{si} \cdot \rho_i$$

3 The Clustering Problem

The common definition of the clustering problem is as follows. Partition n given data points into k clusters, such that points within a cluster are more similar to each other than ones taken from different clusters. The N data points are specified either in term of their coordinates in a d-dimensional space or by means of an $n \times n$ *similarity matrix*, whose elements s_{ij} measure the similarity of data points i and j.

Our algorithms use the similarity matrix only, and thus can deal with cases where pairwise similarity is the only information available about the data. Specifically, we address the problem of clustering the weighted graph $G(V, E, w)$. Most often we prefer to model the data using sparse graphs, which contain only a small subset of the edges of the complete graph, those corresponding to higher similarity values. Working with sparse graphs has several advantages. First, it reduces the time and space complexity, and second, the "structure" of adequate sparse graphs expresses the arrangement of the data, thus helping the clustering process.

A preferred quality of a clustering algorithm is its ability to determine the number k of natural clusters. In reality, however, most clustering algorithms require this number to be an input, which means that they may break up or combine natural clusters, or even create clusters when no natural ones exist in the data.

The problem as described above is inherently ill-posed, since a set of points can be clustered naturally in many ways. For example, Figure 1(a) has three clusters, but one could argue that there are only two, since the two on the right hand side are close enough to be thought of as one. In Figure 1(b) one could argue for and against dividing the points in the top dense region into two highly connected natural clusters. A solution to such ambiguities is to use hierarchical clustering, which employs a parameter for controlling the desired resolution.

Various cost functions, sometimes called objective functions, have been proposed in order to measure the quality of a given clustering. Viewing the clustering problem as an optimization problem of such an objective function formalizes the problem to some extent. However, we are not aware of any function that optimally captures the notion of a 'good' cluster, since for any function one can exhibit cases for which it fails. Furthermore, not surprisingly, no polynomial-time algorithm for optimizing such cost functions is known. In fact, a main role of cost

[1] Notice that when multiplying each row i with d_i, the weighted degree of the respected node, the system is represented with a symmetric positive-definite matrix, which is easier to be solved

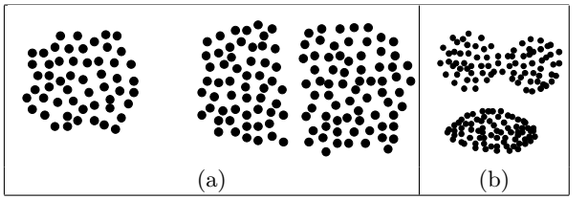


Fig. 1. Inherent ambiguity in clustering: How many clusters are there here?

functions for clustering is to obtain some intuition about the desired properties of a good clustering, and to serve as an objective metric for distinguishing a good clustering from a bad one.

3.1 Clustering Methods

We now survey some clustering approaches. Instead of providing specific references to each method, we point the reader to the surveys in [7,8].

Clustering methods can be broadly classified into *hierarchical* and *partitional* approaches. Partitional clustering algorithms obtain a single partition of the data that optimizes a certain criterion. The most widely used criterion is minimizing the overall squared distance between each data point and the center of its related cluster. This tends to work well with isolated and compact clusters. The most common methods of this kind are the *k-Means* (that is suitable only for points in a metric space) and the *k-Medoid* algorithms. An advantage of these algorithms is their robustness to outliers (nodes that cannot be classified into a natural cluster). Another advantage is their quick running time. Their major drawback is a tendency to produce spherically shaped clusters of similar sizes, which often prevents the finding of natural clusters. For example, consider the graph in Figure 2. A natural clustering decomposition of this graph is into two rectangular grids, the larger left-hand-side grid and the smaller right-hand-side grid. However, these methods will attach some nodes of the left-hand-side grid to the nodes of the right-hand-side grid, seeking to minimize the distance of each node to the center of its related cluster.

Hierarchical algorithms create a sequence of partitions in which each partition is nested into next partition in the sequence. *Agglomerative clustering* is a well-known hierarchical clustering method that starts from the trivial partition of n points into n clusters of size 1 and continues by repeatedly merging pairs of clusters. At each step the two clusters that are most similar are merged, until the clustering is satisfactory. Different similarity measures between clusters result in different agglomerative algorithms. The most widely used variants are the *Single-Link* and the *Complete-Link* algorithms. In Single-Link clustering similarity between clusters is measured as the similarity between the most similar pair of elements, one from each of the clusters, while in Complete-Link clustering the similarity is measured using the least similar pair of elements.

The Complete-Link algorithm tends to break up a relatively large (though natural) cluster into two (unnatural) clusters, and will face similar difficulties to the partitional algorithms discussed above. The Single-Link algorithm has a different problem — the "chaining effect": it can be easily fooled by outliers, merging two clusters that are connected by a narrow string of points. For example, when activated on the graph of Figure 2, it will fail, since the distance between the left-hand-side grid and the right-hand-side grid, is equal to the distance between any two adjacent subgraphs.

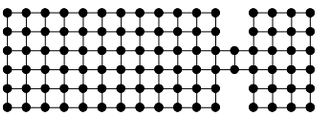


Fig. 2. A natural clustering decomposition of this graph is to divide it into two clusters: the left-hand-side larger grid and the right-hand-side smaller grid. The two nodes connecting these grids are outliers. Our clustering method reveals this decomposition, unlike many traditional clustering methods that will not discover it.

4 Cluster Analysis by Random Walks

4.1 Cluster Quality

Our work is motivated by the following predicate, with which we would like to capture a certain notion of the quality of a cluster:

Definition 4.1 *A cluster C is* (d, α)-normal *iff for every $i, j \in C$ for which $dist(i, j) \leq d$, the probability that a random walk originating at i will reach j before it visits some node outside C, is at least α.*

The role of α is obvious. The reason for limiting the distance between i and j to some d is that for clusters with a large enough diameter it may be easier to escape out of the cluster than to travel between distant nodes inside the cluster. This demonstrates the intuition that in a natural cluster, we need not necessarily seek a tight connection between *every* two nodes, but only between ones that are close enough. For example, consider Figure 2. Random walks starting at nodes in the right-hand-side of the of the large cluster, will probably visit close nodes of the other cluster, before visiting distant nodes of their own cluster.

In fact, the normality predicate can be seen to define the intuitive notion of discontinuities in the data. Such discontinuities indicate the boundaries of the clusters, and are created by sharp local changes in the data.

The normality predicate may label as good the clusters in different clustering decompositions of the same graph. This may be important in some cases,

like when we want to identify a hierarchical clustering decomposition. A disadvantage of this predicate is that when a cluster is not well separated from its neighborhood, the normality predicate may fail to declare the cluster as natural, even though its global structure might be very natural. For example consider Figure 3. A natural clustering decomposition of this graph is to separate the left-hand-side and the right-hand-side grids. However, the normality predicate will not necessarily label these two clusters normal, since there is a relatively wide portion connecting them.

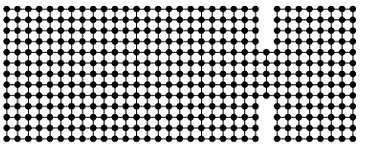


Fig. 3. Giving the normality predicate a hard time

Having said all this, we note that we do not have an efficient method for finding a clustering decomposition that adheres exactly to the normality predicate. However, the algorithms we have developed were conceived of to adhere to its spirit.

4.2 Separators and Separating Operators

Our approach to identifying natural clusters in a graph is to find ways to compute an 'intimacy relation' between the nodes incident to each of the graph's edges. In other words, we want to be able to decide for each edge if it should cross the boundaries of two clusters (when a discontinuity is revealed), or, rather, if the relationship between its two incident nodes is sufficiently intimate for them to be contained in a common cluster.

Definition 4.2 *Let the graph $G(V, E)$ be clustered by $C = (C_1, \ldots, C_k)$. An edge $\langle u, v\rangle \in E$ is called a* separating edge for C*, or a* separator *for short, if $u \in C_i, v \in C_j$ for $i \neq j$.*

Any set of edges $F \subset E$ gives rise to an induced clustering C_F, obtained by simply taking the clusters to be the connected components of the graph $G(V, E - F)$. The set F will then contain precisely the separating edges of C_F. Another way of putting this is that if we can indeed decide which are the separators of a natural clustering of G, we are done, since we will simply take the clustering to be C_F for the discovered set F of separators.

We have decided to concentrate on discovering a set of separating edges, since, in the context of the normality predicate, the decision as to whether an edge should be separating involves only relatively local considerations. Globally

speaking, there might not be much difference between two neighboring nodes, and the reasons for placing two neighbors in different clusters will most often be local. Our philosophy, therefore, is that after identifying the separators by local considerations, we will deduce the global structure of the clustering decomposition by solving an easy global problem of finding connected components.

The strategy we propose for identifying separators is to use an iterative process of *separation*. Separation reweights edges by local considerations in such a way that the weight of an edge connecting 'intimately related' nodes is increased, and for others it is decreased. This is a kind of *sharpening pass*, in which the edges are reweighted to sharpen the distinction between (eventual) separating and non-separating edges. When the separating operation is iterated several times, a sort of 'zero-one' phenomenon emerges, whereby the weight of an edge that should be a separator notably diminishes.

We now offer two methods for performing the edge separation, both based on deterministic analysis of random walks.

NS: Separation by neighborhood similarity. A helpful property of the vector $P^k_{visit}(i)$ is that it provides the level of nearness or intimacy between the node i and every other node, based on the structure of the graph. Actually, $P^k_{visit}(i)$ generalizes the concept of weighted neighborhoods, since $P^1_{visit}(i)$ is exactly the weighted neighborhood of i. Also, $P^\infty_{visit}(i)$ does not depend on i and is equal to the stationary distribution of the graph (when it exists). Hence, the value of $P^k_{visit}(i)$ is not very interesting for overly large values of k. We will actually be using the term $P^{\leq k}_{visit}(\cdot)$, which is defined to be $\sum_{i=1}^{k} P^i_{visit}(v)$.

Now, in order to estimate the closeness of two nodes v and u, we fix some small k (e.g., $k = 3$) and compare $P^{\leq k}_{visit}(v)$ and $P^{\leq k}_{visit}(u)$. The smaller the difference the greater the intimacy between u and v. The reason we use $P^{\leq k}_{visit}$ here and not P^k_{visit} is that for a bipartite subgraph the values of P^k_{visit} can be very different, since the two random walks originating from u and v cannot visit the same node at the same time. However, if we are willing to sum some steps of the two walks, we may find that they visit roughly the same nodes.

We now define the separating operator itself:

Definition 4.3 *Let $G(V, E, w)$ be a weighted graph and k be some small constant. The* separation of G by neighborhood similarity*, denoted by $NS(G)$, is defined to be:*

$$NS(G) \stackrel{dfn}{=} G_s(V, E, w_s),$$
$$where \quad \forall \langle v, u \rangle \in E, \quad w_s(u, v) = sim^k(P^{\leq k}_{visit}(v), P^{\leq k}_{visit}(u))$$

$sim^k(\boldsymbol{x}, \boldsymbol{y})$ is some similarity measure of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, whose value increases as $\boldsymbol{x}$ and $\boldsymbol{y}$ are more similar. A suitable choice is:

$$f^k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{dfn}}{=} \exp(2k - \|\boldsymbol{x} - \boldsymbol{y}\|_{L_1}) - 1$$

The norm L_1 is defined in the standard way: For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$, $\|\boldsymbol{a} - \boldsymbol{b}\|_{L_1} = \sum_{i=1}^{n} |a_i - b_i|$

Another suitable choice is the cosine, or the correlation, of $\boldsymbol{x}$ and $\boldsymbol{y}$ that is defined as:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{(\boldsymbol{x}, \boldsymbol{y})}{\sqrt{(\boldsymbol{x}, \boldsymbol{x})} \cdot \sqrt{(\boldsymbol{y}, \boldsymbol{y})}}$$

where $(\cdot, \cdot)$ denotes inner-product.

The key component in computing $NS(G)$ is the calculation of $P_{visit}^{\leq k}(v)$ and $P_{visit}^{\leq k}(u)$. If the graph G is of bounded degree, $P_{visit}^{\leq k}(u)$ can be computed in time and space $O(deg(G)^k)$, which is independent of the size of G and can be treated as a constant. Hence, for bounded degree graphs $NS(G)$ can be computed in space $O(1)$ and time $\Theta(|E|)$, which in this case is just $\Theta(|V|) = \Theta(n)$.

CE: Separation by circular escape. An alternative method for capturing the extent of intimacy between nodes u and v, is by the probability that a random walk that starts at v visits u exactly once before returning to v for the first time. (This notion is symmetric, since the event obtained by exchanging the roles of v and u has the same probability.) If v and u are in different natural clusters, the probability of such an event will be low, since a random walk that visits v will likely return to v before reaching u (and the same with u and v exchanged).

The probability of this event is given by:

$$P_{escape}(v, u) \cdot P_{escape}(u, v)$$

Seeking efficient computation, and on the reasonable assumption that data relevant to the intimacy of v and u lies in a relatively small neighborhood around v and u, we can constrain our attention to a limited neighborhood, by the following:

Definition 4.4 *Let $G(V, E, w)$ be a graph, and let k be some constant. Denote by $P_{escape}^{(k)}(v, u)$ the probability $P_{escape}(v, u)$, but computed using random walks on the subgraph $G(V^k(\{v, u\}))$ instead of on the original graph G. The* circular escape probability of v and u *is defined to be:*

$$CE^k(v, u) \stackrel{dfn}{=} P_{escape}^{(k)}(v, u) \cdot P_{escape}^{(k)}(u, v).$$

We can now define separation by circular escape:

Definition 4.5 *Let $G(V, E, w)$ be a weighted graph, and let k be some small constant. The* separation of G by circular escape*, denoted by $CE(G)$, is defined to be:*

$$\begin{aligned} &CE(G) \stackrel{dfn}{=} G_s(V, E, w_s) \\ &\textit{where} \quad \forall \langle v, u \rangle \in E, \quad w_s(u, v) = CE^k(v, u) \end{aligned}$$

For graphs with bounded degree, the size of $G(V^k(v, u))$ is independent of the size of G, so that $CE^k(v, u)$ can be computed essentially in constant time and space. Hence, as with $NS(G)$, the separating operator $CE(G)$ can be computed in time $\Theta(|E|) = \Theta(n)$ and space $O(1)$.

4.3 Clustering by Separation

The idea of separating operators is to uncover and bring to the surface a closeness between nodes that exists implicitly in the structure of the graph. Separating operators increase the weights of intra-cluster edges and decrease those of inter-cluster ones. Iterating the separating operators sharpens the distinction further. After a small number of iterations we expect the difference between the weights of the two kinds of edges to differ sufficiently to be readily apparent, because the weights of separators are expected to diminish significantly.

The partition of the edges into separators and non-separators is based on a threshold value, such that all the edges whose weight is below this value are declared as separators. Without loss of generality, we may restrict ourselves to the $O(|E|)$ edge weights as candidates for being thresholds. The actual threshold value (or several, if a hierarchy of decompositions is called for), is found by some statistical test, e.g., inspecting the edge-weight frequency histogram, where the frequency of the separators' weights is usually smaller, since most of the edges are inside the clusters, and have higher weights than those of the separators.

We demonstrate this method by several examples. Consider Figure 4, which contains an almost uniformly weighted graph, taken from [12]. We experimented with both separating operators, each one with a four-fold iteration. The NS operator was used with $k = 3$ and $sim^k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{dfn}}{=} f^k(\boldsymbol{x}, \boldsymbol{y})$ and the CE operator with $k = 2$, other choices work very similarly. The results of both runs appear along the edges in the figure (with those of CE appearing, multiplied by 100, in parentheses). As can be seen, the separation iterations cause the weights of edges $\langle 3, 18\rangle$, $\langle 7, 8\rangle$, $\langle 6, 10\rangle$, $\langle 1, 4\rangle$, and $\langle 8, 18\rangle$ to become significantly smaller than those of the other edges; in fact, they tend to zero in a clear way. We conclude that these edges are separators, thus obtaining the natural clustering of the graph by removing them and taking each connected component of the resulting graph to be a cluster, as indicated in the lower right hand drawing of the figure.

Notice that the first activation of the separating operator already shows differences in the intimacy that later lead to the clustering, but the results are not quite sharp enough to make a clear identification of separators. Take edge $\langle 6, 10\rangle$, for example. We intentionally initialized it to be of weight 1.5 — higher than the other edges — and after the first separation its weight is still too high to be labeled a separator. It is still higher than that of the non-separating edge $\langle 10, 13\rangle$. However, the next few iterations of the separating operator cause its weight to decrease rapidly, sharpening the distinction, and its being a separator becomes obvious.

The success in separating nodes 6 and 10 is particularly interesting, and would probably not be possible by many clustering methods. This demonstrates how our separation operators integrate structural properties of the graph, and succeed in separating these nodes despite the fact that the edge joining them has the highest similarity value in the graph.

Figure 5 shows the algorithms applied to a tree, using three-fold separation. The results clearly establish edges $\langle 0, 6\rangle$ and $\langle 6, 11\rangle$ as separators. Notice that

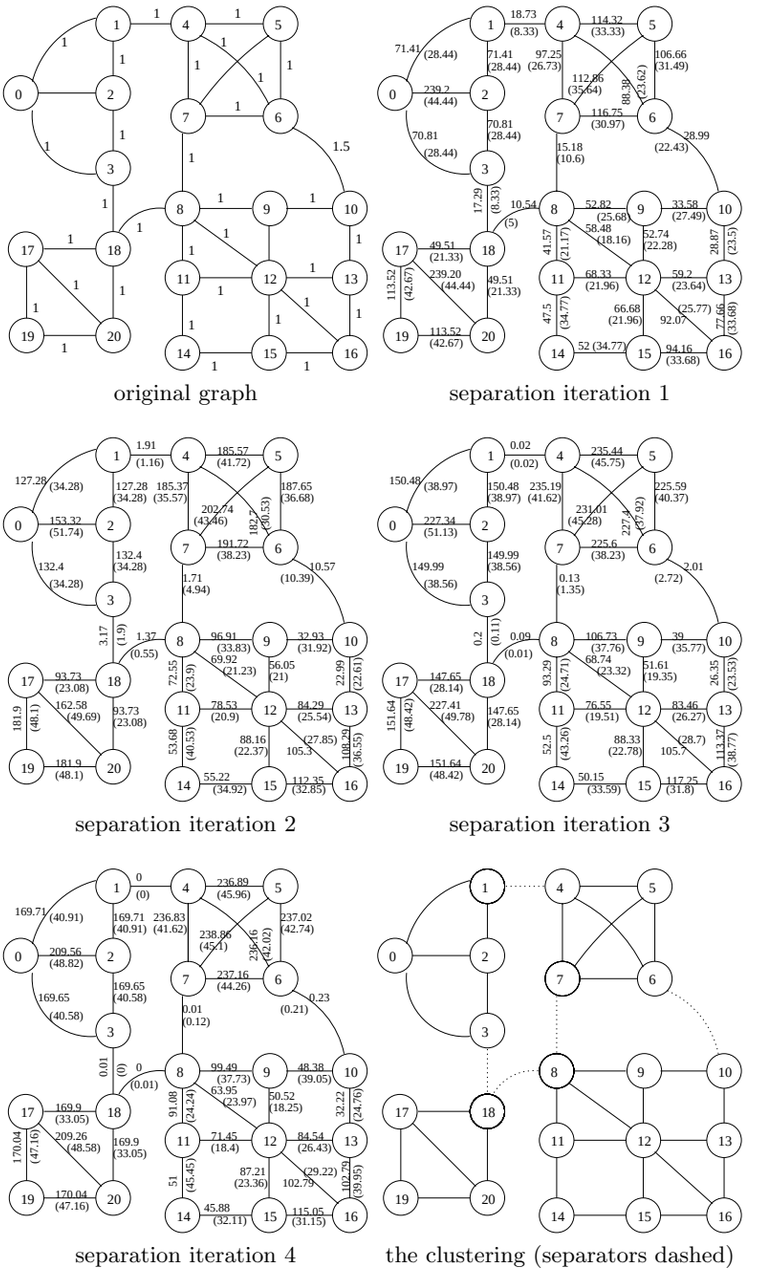


Fig. 4. Clustering using four-fold application of separation operators, which sharpen the edge weight distinction (NS values are on top and CE values, multiplied by 100, are in parentheses); example taken from [11]

the clustering methods that rely on edge-connectivity will fail here, since the edge-connectivity between every two nodes of the tree is one.

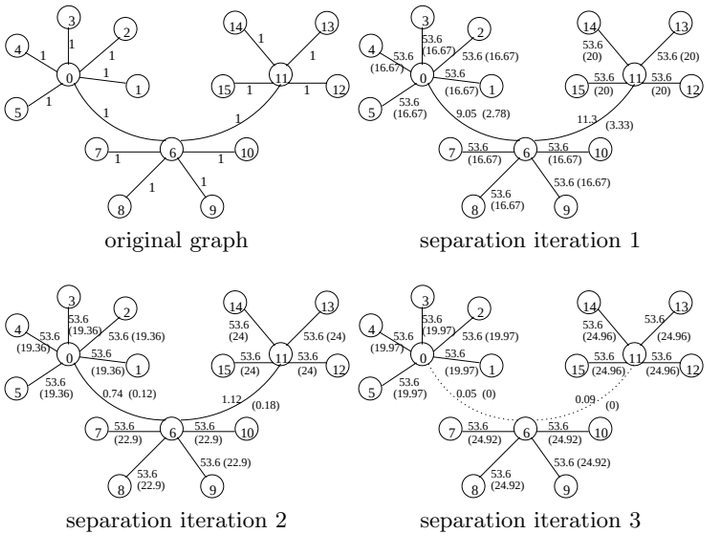
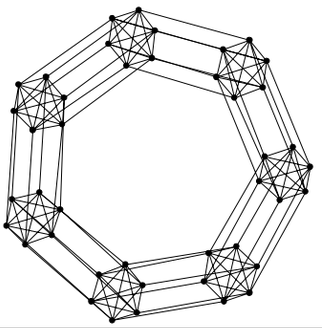


Fig. 5. Clustering a tree using three-fold application of separation (separators are dashed in iteration 3)

Figure 6 shows seven copies of the complete graph K_6 arranged cyclically. Each node is linked by 'internal' edges to the other five nodes in its own complete subgraph, and by 'external' edges to two nodes of the neighboring complete subgraphs. All edges have uniform weight. In the table, one can see how iterations of the separating operators diminish the weights of the 'external' edges, which are clearly found to be separators, decomposing the graph into seven clusters of complete subgraphs.

When applying the separating operators to the graph of Figure 2, the edges of the lowest sharpened weight are those outside the two grids, resulting in the decomposition of the graph into three clusters, as shown in Figure 7.

The graph in Figure 8(a) demonstrates the application of our method to a weighted graph. The weight of edges of this graph have been set up to decrease exponentially with their length. After a three-fold iteration of the CE separating operator with $k = 3$, and declaring the edges with weight below 0.097 as separators, the graph is decomposed into two clusters depicted in Figure 8(b). Slight changes to the value of k, or applying the NS separating operator, produce similar results, where several nodes on the chains connecting the upper and the lower clusters become independent clusters of outliers.



	1	2	3	4	5	6
CE	30.56 / 16.21	33.16 / 9.51	34.51 / 4.74	35.31 / 1.65	35.77 / 0.26	35.96 / 0
NS	191.38 / 12.08	279.17 / 0.33	287.14 / 0.01	287.3 /0	287.3 /0	287.3 / 0

Fig. 6. Clustering a cycle of complete graphs. Edges are of two kinds: internal edges that link two nodes of the same complete subgraph and external edges linking nodes of different subgraphs. The table shows the sharpened weights of internal/external edges after each of six iterations of separation.

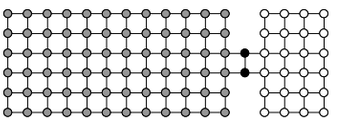
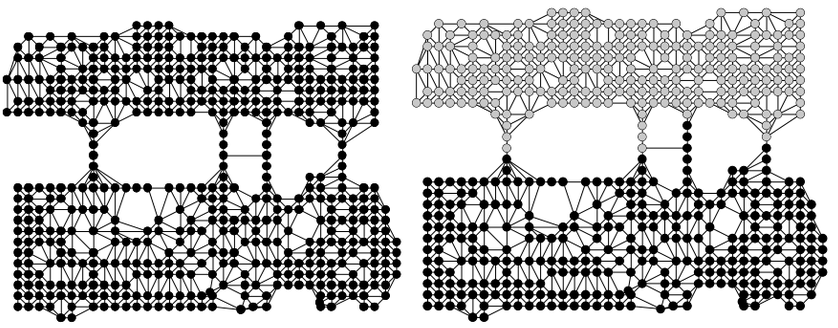


Fig. 7. Clustering the graph of Figure 2. The 3 clusters are denoted by different colors.



(a) (b)

Fig. 8. (a) A weighted graph (edge weights are decaying exponentially with their length); (b) decomposition of the graph into two clusters.

The main virtues of clustering by separation are:

1. Applying a separation operator to an edge in a bounded-degree graph takes constant time and space, resulting in very good time complexity for large bounded-degree graphs.
2. Edges are weighted based on the relevant graph structure, thus overcoming phenomena like random noise and outliers, which are not reflected directly in the structure.
3. Iterating the separating operators causes information from distant parts of the graph to 'flow in', reaching the areas where separating decisions are to be made.

Notice that the differences between consecutive iterations of separation diminish as the process continues, and they appear to tend to a fixpoint. This behavior requires further investigation.

4.4 Clustering Spatial Point-Sets

We now illustrate the ability of our method to cluster "correctly" 2D sets of points, in a number of typical cases, some of which have been shown to be problematic for agglomerative methods [9]. (More extensive examples are given in Subsection 5.1.) For a short version of this paper that deals with clustering spatial data, see [5].

We have used 10-mutual neighborhood graphs for modeling the points. The k-mutual neighborhood graph contains all edges $\langle a, b\rangle$ for which a is one of the k nearest neighbors of b, and b is one of the k nearest neighbors of a. Regarding edge weights, we adopt a commonly used approach: the weight of the edge $\langle a, b\rangle$ is $\exp(-\frac{d(a,b)^2}{ave^2})$, where $d(a, b)$ is the Euclidean distance between a and b, and ave is the average Euclidean distance between two adjacent points in the graph.

The results are achieved using 3 iterations of either CE or NS, with $k = 3$. For NS, we took the function $sim(\cdot, \cdot)$ to be $f(\cdot, \cdot)$. In general, other choices work equally well.

The partition of the edges into separators and non-separators is based on a threshold value, such that all the edges whose weight is below this value are declared as separators. Without loss of generality, we may restrict ourselves to the $O(n)$ edge weights as candidates for being thresholds. The actual threshold value (or several, if a hierarchy of decompositions is called for), is found by some statistical test, e.g., inspecting the edge-weight frequency histogram, where the frequency of the separators' weights is usually smaller, since most of the edges are inside the clusters, and have higher weights than those of the separators.

Figure 9 shows the clustering decomposition of three data sets using our algorithm.

The data set DS1 shows the inherent capability of our algorithms to cluster at different resolutions at once, i.e., to detect several groups with different intra-group densities. This ability is beyond the capabilities of many clustering algorithms that can show the denser clusters only after breaking up the sparser

clusters. Data set DS2 demonstrates the ability of our algorithm to separate the two left hand side clusters, despite the fact that the distance between these clusters is smaller than the distance between points inside the right hand side cluster.

The data set DS3 exhibits the capability of our algorithm to take into account the structural properties of the data set, which is the only clue for separating these evenly spaced points.

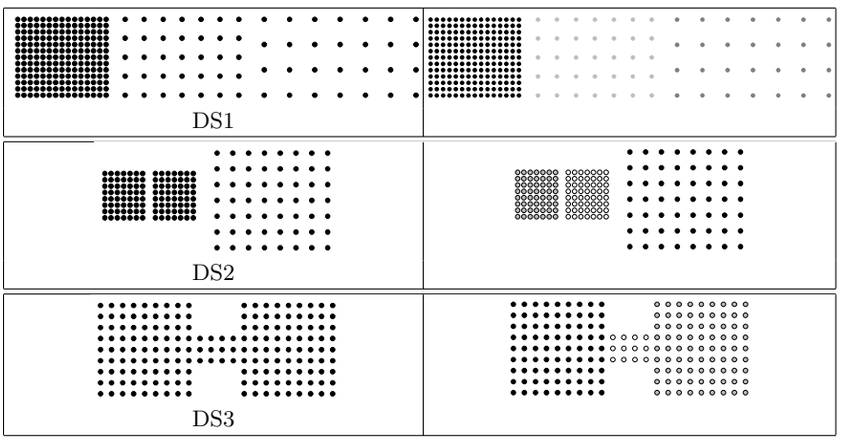


Fig. 9. Clustering of several data sets. Different clusters are indicated by different colors.

When there is a hierarchy of suitable decompositions, our method can reveal it by using a different threshold for each level of the hierarchy. For example, consider the two data sets of Figure 1. For each of these we have used two different thresholds, to achieve two decompositions. The results are given in Figure 10.

5 Integration with Agglomerative Clustering

The separation operators can be used as a preprocessing stage before activating agglomerative clustering on the graph. (Without loss of generality, we think of agglomerative algorithms as working on graphs.) Such preprocessing sharpens the edge weights, adding structural knowledge to them, and greatly enhances the agglomerative algorithms, as it can effectively prevent bad local merging that works against the graph structure.

Implementation of the agglomerative algorithm can be done using a dynamic graph structure. At each step we take the edge of the highest weight, merge ("contract") its two endpoints, and update all the adjacent edges. When contracting nodes u and v having a common neighbor t, the way we determine the

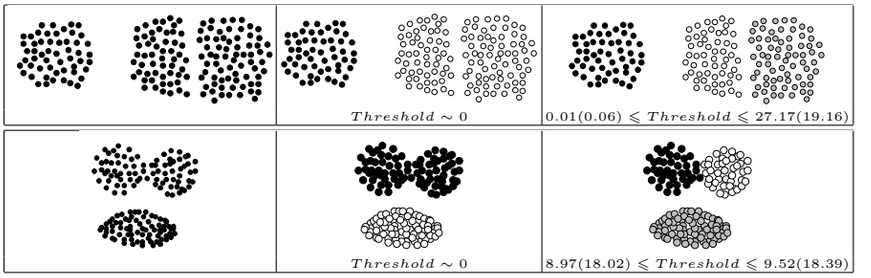


Fig. 10. Clustering at multiple resolutions using different thresholds. When values of CE are different from values of NS, the CE values are given in parentheses. CE values are multiplied by 100.

weight of the edge between t and the contracted node uniquely distinguishes between different variants of the agglomerative procedure. For example, when using Single-Link, we take this weight as $\max\{w(v,t), w(u,t)\}$, while when using total similarity we fix the weight as $w(v,t) + w(u,t)$. For a bounded degree graph, which is our case, each such step can be carried out in time $O(\log n)$, using a binary heap.

It is interesting that the clustering method we have described in the previous section is in fact equivalent to a Single-Link algorithm preceded by a separation operation. Hence we can view the integration of the separation operation with the agglomerative algorithm as a generalization of the method we have discussed in the previous section, which enables us to use any variant of the agglomerative algorithm.

We have found particularly effective the normalized total similarity variant, in which we measure the similarity between two clusters as the total sum of the weights of the original edges connecting these clusters. We would like to eliminate the tendency of such a procedure to contract pairs of nodes representing large clusters whose connectivity is high due to their sizes. Accordingly, we normalize the weights by dividing them by some power of the sizes of the relevant clusters. More precisely, we measure the similarity of two clusters C_1 and C_2 by:

$$\frac{w(C_1, C_2)}{\sqrt[d]{|C_1|} + \sqrt[d]{|C_2|}}$$

where $w(C_1, C_2)$ is the sum of original edge weights between C_1 and C_2, and d is the dimension of the space in which the points lie. We took $\sqrt[d]{|C_1|}$ and $\sqrt[d]{|C_2|}$ as an approximation of the size of the boundaries of the clusters C_1 and C_2, respectively.

The overall time complexity of the algorithm is $O(n \log n)$, which includes the time needed for constructing the graph and the time needed for performing n contractions using a binary heap. This equals the time complexity of the method described in the previous section (because of the graph construction

stage). However, the space complexity is now worse. We need $\Theta(n)$ memory for efficiently handling the binary heap.

Selecting Significant Decompositions

An agglomerative clustering algorithm provides us with a dendrogram, which is a pyramid of nested clustering decompositions. It does not directly addresses the question of which are the meaningful decompositions inside the dendrogram.

Each level in the dendrogram is constructed from the level below, by merging two clusters. We associate with each level a grade that measures the importance of that level. Inspired by the work of [2], a rather effective way of measuring the importance of a level is by evaluating how sharp is the change that this level introduces into the clustering decomposition. Since changes that are involved with small clusters do not have a large impact, we define the *prominency rank* of a level in the dendrogram, in which the clusters C_i and C_j of the level below were merged, as:

$$|C_i| \cdot |C_j|$$

We demonstrate the effectiveness of this measure in the next section.

5.1 Examples

In this section we show the results of running our algorithm on several data sets from the literature. For all the results we have used total similarity agglomerative clustering, preceded by 2 iterations of the NS separation operator with $k = 3$ and similarity function defined as $cos(\cdot, \cdot)$. Using the CE operator, changing the value of k, or increasing the number of iterations, do not have a significant effect on the results. Using the method described in Section 4 may change the results in few cases.

We implemented the algorithm in C++, running on a Pentium III 800MHz processor. The code for constructing the Delaunay triangulation is of Triangle, which is available from URL: http://www.cs.cmu.edu/~quake/triangle.html. The reader is encouraged to see [6], in order to view the figures of this section in color.

Figure 11 shows the results of the algorithm on data sets taken from [9]. These data sets contain clusters of different shapes, sizes and densities and also random noise. A nice property of our algorithm is that random noise gets to stay inside small clusters. After clustering the data, the algorithm treats all the relatively small clusters, whose sizes are below half of the average cluster size, as noise, and simply omits them showing only the larger clusters.

Figure 12 shows the result of the algorithm applied to a data set from [1]. We show two levels in the hierarchy, representing two possible decompositions. We are particularly happy with the algorithm's ability to break the cross shaped cluster into 4 highly connected clusters, as shown in Figure 12(c).

In Figure 13, which was produced by adding points to a data set given in [1], we show the noteworthy capability of our algorithm to identify clusters of different densities at the same level of the hierarchy. Notice that the intra-distance

between the points inside the right hand side cluster, is larger than the inter-distance between several other clusters.

The data set in Figure 14, which in a way is the most difficult one we have included, is taken from [2]. We have modeled the data exactly the same way described in [2], by putting an edge between every two points whose distance is below some threshold. Using this model, [2] shows the inability of two spectral methods and of the Single-Link algorithm to cluster this data set correctly.

Throughout all the examples given in this section, we have used the prominency rank introduced in Section 5 to reveal the most meaningful levels in the dendrogram. Figure 15 demonstrates its capability with respect to the data set DS4 (shown in Figure 11). We have chosen the five levels with the highest prominency ranks, and for each level we show the level that precedes it. It can be seen that these five levels are exactly the five places where the six large natural clusters are merged. In this figure we have chosen not to hide the noise, so the reader can see the results of the algorithm before hiding the noise.

Table 1 gives the actual running times of the algorithm on the data sets given here. We should mention that our code is not optimized, and the running time can certainly be improved.

Table 1. Running time (in seconds; non-optimized) of the various components of the clustering algorithm

Data Set	Size	Graph construction	Separation	Agglomeration	Overall	Ratio $\frac{Points}{Sec}$
DS4	8000	0.4	0.88	0.19	1.47	5434
DS5	8000	0.41	0.83	0.19	1.43	5587
DS6	10000	0.5	1.12	0.26	1.88	5311
DS7	8000	0.4	0.89	0.2	1.49	5358
DS8	8000	0.39	0.93	0.2	1.52	5256
DS9	8000	0.33	0.66	0.21	1.2	6656
DS10	3374	0.14	0.26	0.07	0.47	7178

6 Multi-scale Clustering

In this section we embed our separation operators inside a classical multi-scale scheme. The resulting algorithm subsumes the agglomerative variant presented in Section 5.

A multi-scale treatment of a graph-related problem handles the graph in a global manner, by constructing a *coarse abstraction* thereof. The abstraction is a new graph that contains considerably fewer nodes than the original, while preserving some of its crucial properties. Dealing with a global property for the coarse graph may be easier, since it contains much less information, and hopefully still has the desired property. A *multi-scale* representation of the graph consist of various coarse abstractions that allow us to view the graph on different scales, that differ in the level of abstraction they represent. For example, see [10,11].

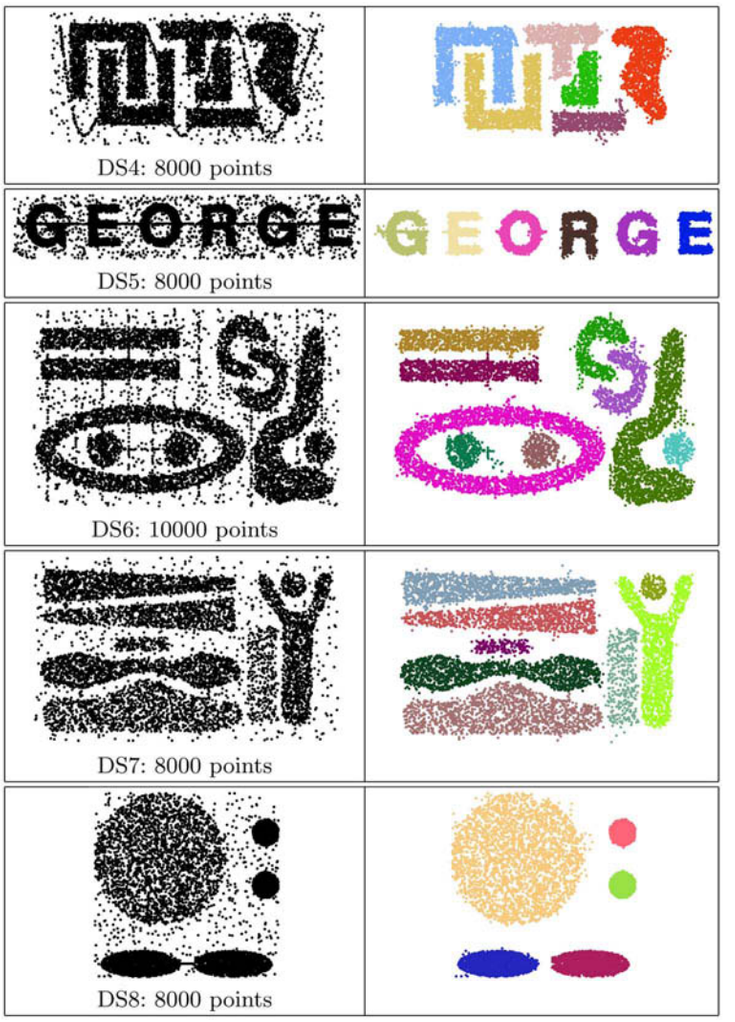


Fig. 11. Data sets taken from [9] (see [6] for clearer color versions of this figure and of Figs. 12–15).

6.1 The General Scheme

In our context, we find that the multi-scale technique is often called for in order to identify clusters whose naturalness stems from the graph's global structure, and which would be very difficult to identify using only local considerations. Such clusters are not well separated from their surroundings. For example, there might be wide 'channels of leaked information' between such a cluster and others, disrupting separation. If we were able to construct a coarse graph in which a wide

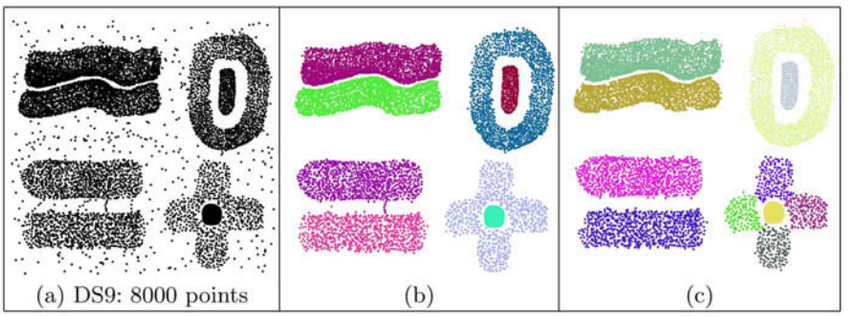


Fig. 12. Two different clusterings of a data set taken from [1]

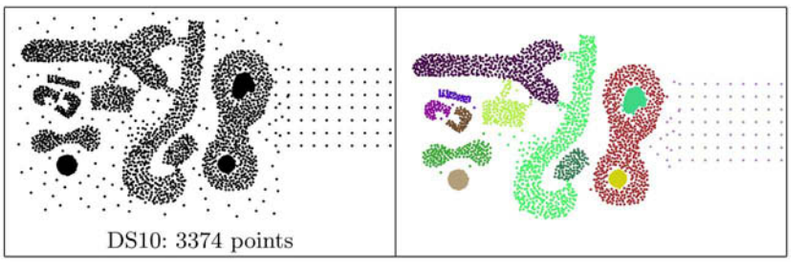


Fig. 13. A data set with clusters of different densities

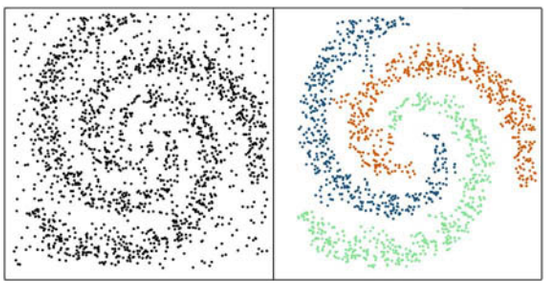


Fig. 14. A 2012 points data set taken from [2]

channel of connection is replaced by a single separating edge, we would be able to overcome the difficulty.

For example, consider the graph in Figure 3. As mentioned earlier, the natural clustering decomposition of this graph does not obey the predicate for cluster normality introduced in Section 4.1, due to the broad five-edge connection between the two larger parts of the graph. Hence, our separating operators, which were developed in the spirit of this predicate, will have a hard time identifying the natural decomposition of this graph. (The operators do manage to natu-

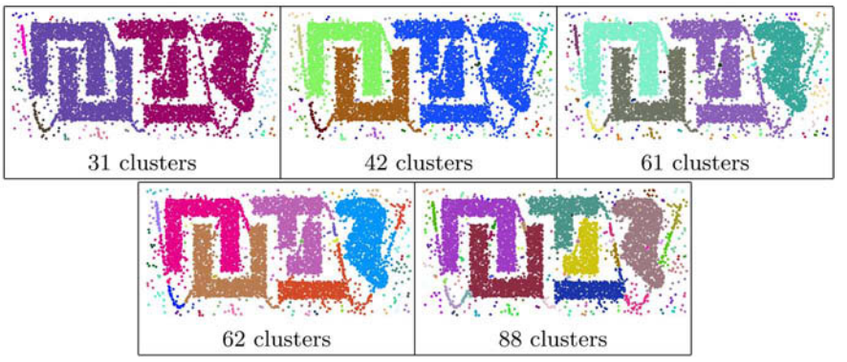


Fig. 15. A hierarchy containing five decompositions of DS4 corresponding to the five levels with the highest prominency rank.

rally decompose this graph if they applied with a relatively large value of k, i.e., $k > 5$.) The multi-scale solution we propose below overcomes this situation by constructing a coarse representation similar to that of Figure 2, in which the natural decomposition is correctly identified. We can then use the decomposition of the coarse graph to cluster the original one, as long as we have a good way of establishing a correspondence between the left and right grids of the two graphs, respectively.

Here, now, is a high-level outline of the multi-scale clustering algorithm.

MS-Clustering $(G(V, E, w))$

1. compute iterated sharpened weights of G's edges.
2. *if* G is small enough *then*
   return a clustering decomposition of G.
3. construct $G^C(V^C, E^C, w^C)$, a coarser abstraction of G, such that $|V^C| = \alpha \cdot |V|$, where $0 < \alpha < 1$.
4. call **MS-Clustering**$(G^C(V^C, E^C, w^C))$.
5. obtain a clustering of G by projecting the clustering of G^C onto G.
6. improve the clustering of G using a greedy smoothing procedure.
7. *end*

6.2 Structure Preserving Coarsening

Clearly, the key step in the algorithm is the computation of a coarse graph, which we now set out to describe. A common approach to coarsening graphs is to use a series of *edge-contractions*. In a single operation of edge-contraction we pick some edge $\langle v, u \rangle$, and combine the two nodes v and u ('fine nodes') into a single super-node $v \cup u$ ('coarse-node'). In order to preserve the connectivity information in the coarse graph, we take the set of edges of $v \cup u$ to be the union

of the sets of edges of v and u. If v and u have a common neighbor t, the weight of the edge $\langle v \cup u, t \rangle$ is taken to be $w(v,t) + w(u,t)$.

A coarse graph is only useful to us if it retains the information related to the natural clustering decomposition of the original graph. Hence, we seek what we call a *structure preserving coarsening* in which large-enough natural clusters of the original graph are preserved by the coarsening. A key condition for this is that a coarse node does not contain two fine nodes that are associated with different natural clusters; or, equivalently, that we do not contract a separating edge.

To achieve this, we select the edges to be contracted by considering the sharpened weights of the edges — those obtained by using our separating operators — and contract only edges with high sharpened weights. We would like to eliminate the tendency of such a procedure to contract pairs of large nodes whose connectivity is high due to their sizes. Accordingly, we normalize the sharpened weights by dividing them by some power of the sizes of the relevant nodes.

The hope is that the kind of wide connections between natural clusters that appear in Figure 3 will show up as sets of separators. This is based on the fact that connections between sets of nodes that are related to the same cluster should be stronger than connections between sets that are related (even partially) to different clusters.

After finding the clustering decomposition of the coarse graph, we deduce the related clustering decomposition of the original graph by a simple projection based on the inclusion relation between fine and coarse nodes. The projected clustering might need refining: When the wide connections indeed exist, it may be hard to find the 'right' boundary of a natural cluster, and some local mistakes could occur during coarsening. We eliminate this problem by adding a *smoothing* phase (line 6 of the algorithm), in which we carry out an iterative greedy process of exchanging nodes between the clusters. The exchanges are done in such a way that each node joins the cluster that minimizes some global cost-function (we have chosen the multi-way cut between all the clusters). This kind of smoothing is similar to what is often done in graph partitioning; see, e.g., [10].

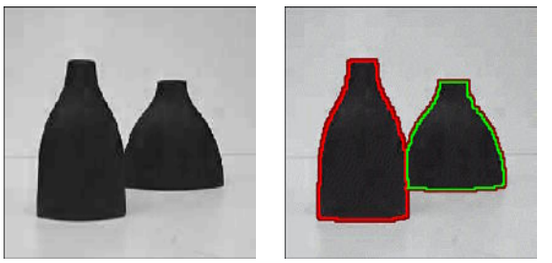
6.3 Relationship to Agglomerative Clustering

The edge contraction operation in our multi-scale clustering method is essentially the same as the merging of two clusters in agglomerative algorithms.

The main difference between the multi-scale method and the agglomerative variant introduced in Section 5, is in the use of the separating operators on the coarse graphs, not only on the original fine graph: at certain levels of the process we sharpen the edge weights by these operators. This way, the operators act upon more global properties, which can be identified on the coarser graphs. Another advantage of the multi-scale algorithm, is its utilization of the smoothing process, which can undo erroneous merges.

We have found this multi-scale algorithm superior for the task of image segmentation. A common approach to image segmentation is to represent the image by a weighted graph whose nodes are the pixels of the image. There are edges

connecting each pixel with its four immediate neighbors, and the weight of an edge is determined by the similarity of the intensity levels of the incident pixels. Figure 16 shows the ability of the multi-scale algorithm to accurately separate the two vases, in spite of the large connectivity between them. We are still in the process of investigating the use of our ideas in image segmentation, and we expect to present additional results.



(a) (b)

Fig. 16. (a) original 350×350 image (taken from [10]); (b) segmentation: each vase forms its own segment

7 Related Work

Random walks were first used in cluster analysis in [4]. However, the properties of the random walk there are not computed deterministically, but by a random algorithm that simulates a $\Theta(n^2)$ random walk. This results in time and space complexity of $\Theta(n^3)$ and $\Theta(n^2)$, respectively, even on bounded degree graphs.

A recent algorithm that uses deterministic analysis of random walks for cluster analysis is that of [13]. The approach there is quite different from ours. Also, its time and space complexity appear to be $\Omega(n^3)$ and $\Theta(n^2)$, respectively, even for bounded degree graphs.

A recently published graph-based approach to clustering, aimed at overcoming the limitations of agglomerative methods, is [9]. It is hard for us to assess its quality since we do not have its implementation. However, the running time of [9], which is $O(nm + n \log n + m^2 \log m)$ for $m \sim 0.03n$, is slower than ours.

Finally, we mention [3], in which an agglomerative clustering algorithm is described that merges the two clusters with the (normalized) greatest number of common neighbors. To our best knowledge, this is the first agglomerative algorithm that considers properties related directly to the structure of the graph. Our work can be considered to be a rather extensive generalization of this work, in the sense that it considers weights of edges and adds considerations related to larger neighborhoods.

8 Conclusion

The process of using random-walk-based separating operators for clustering seems to have a number of major advantages. One advantage is in the quality of the resulting clustering. Our algorithms can reveal clusters of any shape without a special tendency towards spherically shaped clusters or ones of similar sizes (unlike many clustering algorithms that tradeoff these features for being robust against outliers). At the same time, the decisions the algorithms make are based on the relevant structure of the graph, making them essentially immune to outliers and noise.

Another advantage is the running time. Our separating operators can be applied in linear time when the graphs are of bounded degree, and their running time in general is very fast. We have been able to cluster 10,000-node planar graphs in less than two seconds, on a 700MHz Pentium III PC.

In addition to developing the algorithms themselves, we have also attempted to define a rather general criterion for the naturalness of a cluster (Section 4.1). We hope to use this criterion in the future as the basis of improved algorithms, and to better study the connections between it and random-walk-based separation.

Finally, we believe that the structure preserving coarsening introduced in Section 6 can be used to improve other algorithms that perform coarsening on structured graphs, e.g., multi- scale graph drawing algorithms and multi-level graph partitioning algorithms [10].

References

1. V. Estivill-Castro and I. Lee,"AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point- Data Sets", *5th International Conference on Geocomputation*, GeoComputation CD-ROM: GC049, ISBN 0-9533477-2-9.
2. Y. Gdalyahu, D. Weinshall and M. Werman, "Stochastic Image Segmentation by Typical Cuts", *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 1999, pp. 588–601.
3. S. Guha, R. Rastogi and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes", *Proceedings of the 15th International Conference on Data Engineering*, pp. 512–521, 1999.
4. L. Hagen and A. Kahng, "A New Approach to Effective Circuit Clustering", *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 422–427, 1992.
5. D. Harel and Y. Koren, "Clustering Spatial Data using Random Walks", *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (KDD-2001), ACM, pp. 281–286, 2000.
6. D. Harel and Y. Koren, "Clustering Spatial Data Using Random Walks", Technical Report MCS01-08, Dept. of Computer Science and Applied Mathematics, The Weizmann Institute of Science, 2001. Available at: `www.wisdom.weizmann.ac.il/reports.html`
7. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, New Jersy, 1988.

8. A. K. Jain, M.N. Murty and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys*, **31** (1999), 264–323.
9. G. Karypis, E. Han, and V. Kumar, "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling", *IEEE Computer*, **32** (1999), 68–75.
10. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing* **20**:1 (1999), 359–392.
11. E. Sharon, A. Brandt and R. Basri, "Fast Multiscale Image Segmentation", *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pp. 70–77, 2000.
12. B. Stein and O. Niggemann, "On the Nature of Structure and its Identification", *Proceedings 25th Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 1665, pp. 122–134, Springer Verlag, 1999.
13. N. Tishby and N. Slonim, "Data Clustering by Markovian relaxation and the Information Bottleneck Method", *Advances in Neural Information Processing Systems 13*, 2000.

An Introduction to Decidability of DPDA Equivalence

Colin Stirling

Division of Informatics
University of Edinburgh
cps@dcs.ed.ac.uk

1 Introduction

The DPDA equivalence problem was posed in 1966 [4]: is there an effective procedure for deciding whether two configurations of a deterministic pushdown automaton (a DPDA) accept the same language? The problem is whether language equivalence is decidable for deterministic context-free languages. Despite intensive work throughout the late 1960s and 1970s, the problem remained unsolved until 1997 when Sénizergues announced a positive solution [11]. It seems that the notation of pushdown configurations, although simple, is not rich enough to sustain a proof. Deeper algebraic structure needs to be exposed. The full proof by Sénizergues, in journal form, appeared earlier this year [12]. It exposes structure within a DPDA by representing configurations as boolean rational series, and he develops an algebraic theory of their linear combinations. Equivalence between configurations is captured within a deduction system. The equations within the proof system have associated weights. Higher level strategies (transformations) are defined which guide proof. A novel feature is that these strategies depend upon differences between weights of their associated equations. Decidability is achieved by showing that two configurations are equivalent if, and only if, there is a finite proof of this fact.

I produced a different proof of decidability that is essentially a simplification of Sénizergues's proof [14]. It is based on a mixture of techniques developed in concurrency theory and language theory. The first step is to view the DPDA problem as a bisimulation equivalence problem for a process calculus whose expressions generate infinite state transition systems. The process calculus is built from determinising strict grammars: strict grammars were introduced by Harrison and Havel [5] because they are equivalent to DPDA. Tableaux proof systems have been used to show decidability of bisimulation equivalence between infinite state processes, see, for instance, [8,2]. I use this method for the DPDA problem. However, the tableau proof system uses conditional proof rules that involve distances between premises. Essentially this is Sénizergues's use of weights, and the idea was developed from trying to understand his proof.

The proof of decidability is unsatisfactory. It is very complex because the proof of termination uses a mechanism for "decomposition" that in [14] is based on unifiers and auxiliary recursive nonterminals (from [3,13]). Sénizuergues uses

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 42–56, 2001.
© Springer-Verlag Berlin Heidelberg 2001

a more intricate mechanism. This means that the syntax of the starting process calculus has to be extended in [14] with auxiliary symbols. It also introduces nondeterminism into tableaux with the consequence that the decision procedure (in both [12,14]) is two semidecision procedures. The result is that there is no known upper bound on complexity.

In this paper I describe a simpler decision procedure that should lead to an elementary complexity upper bound. It is a deterministic procedure that avoids the decomposition mechanism for termination (the rule CUT in [14] and the transformation T_C in [12]). Instead, there is a new and much simpler analysis of termination. It also means that the syntax of the starting process calculus is not extended. The paper is entirely introductory, and contains no proofs. Section 2 introduces the DPDA problem as a bisimulation equivalence problem. Section 3 describes some features of the process calculus in more detail. Finally, Section 4 introduces the deterministic tableau proof decision procedure. The aim of the paper is to provide the reader with a clear indication of the decision procedure: more details, and full proofs, can be found at the author's home page.

2 DPDA and Strict Grammars

A deterministic pushdown automaton, a DPDA, consists of finite sets of states P, stack symbols S, alphabet A and basic transitions T. A basic transition is $pS \stackrel{a}{\longrightarrow} q\alpha$ where p, q are states in P, $a \in \mathsf{A} \cup \{\varepsilon\}$, S is a stack symbol in S and α is a sequence of stack symbols in S^*. Basic transitions are restricted.

$$\text{if } pS \stackrel{a}{\longrightarrow} q\alpha \in \mathsf{T} \text{ and } pS \stackrel{a}{\longrightarrow} r\beta \in \mathsf{T} \text{ and } a \in \mathsf{A} \cup \{\varepsilon\}, \text{then } q = r \text{ and } \alpha = \beta$$
$$\text{if } pS \stackrel{\epsilon}{\longrightarrow} q\alpha \in \mathsf{T} \text{ and } pS \stackrel{a}{\longrightarrow} r\lambda \in \mathsf{T} \text{ then } a = \epsilon$$

A configuration of a DPDA has the form $p\delta$ where $p \in \mathsf{P}$ is a state and $\delta \in \mathsf{S}^*$ is a sequence of stack symbols. The transitions of a configuration are determined by the following prefix rule, assuming that $\beta \in \mathsf{S}^*$: if $pS \stackrel{a}{\longrightarrow} q\alpha \in \mathsf{T}$ then $pS\beta \stackrel{a}{\longrightarrow} q\alpha\beta$.

The transition relation $\stackrel{a}{\longrightarrow}$, $a \in \mathsf{A} \cup \{\varepsilon\}$, between configurations is extended to words $\stackrel{w}{\longrightarrow}$, $w \in \mathsf{A}^*$. First, $p\alpha \stackrel{\epsilon}{\longrightarrow} p_n\alpha_n$, if $p_n = p$ and $\alpha_n = \alpha$ or there is a sequence of basic transitions $p\alpha \stackrel{\epsilon}{\longrightarrow} p_1\alpha_1 \stackrel{\epsilon}{\longrightarrow} \ldots \stackrel{\epsilon}{\longrightarrow} p_n\alpha_n$. If $w = av \in \mathsf{A}^+$, then $p\alpha \stackrel{w}{\longrightarrow} q\beta$ if $p\alpha \stackrel{\epsilon}{\longrightarrow} p'\alpha' \stackrel{a}{\longrightarrow} q'\beta' \stackrel{v}{\longrightarrow} q\beta$. A configuration $pS\alpha$ is either "stable" and has no ε-transitions or it is "unstable" and only has a single ε-transition. We assume that a final configuration $p\varepsilon$ with the empty stack is also stable. Clearly, if $p\alpha \stackrel{w}{\longrightarrow} q\beta$ and $p\alpha \stackrel{w}{\longrightarrow} r\delta$ and $q\beta$ and $r\delta$ are stable, then $q = r$ and $\beta = \delta$. The language accepted, or generated, by a configuration $p\delta$, written $\mathsf{L}(p\delta)$, is the set of words $\{w \in \mathsf{A}^* \ : \ \exists q \in \mathsf{P}.\, p\delta \stackrel{w}{\longrightarrow} q\epsilon\}$. That is, acceptance is by empty stack and not by final state. The motivation for providing a positive solution to the decision problem is to establish decidability of language equivalence between deterministic context-free languages. However, DPDAs which have empty stack acceptance can only recognise the subset of deterministic context-free languages that are prefix-free: a language L is prefix-free if $w \in L$, then no proper prefix of w is also in L. However, DPDAs whose

acceptance is by final state do recognise all deterministic context-free languages. A DPDA with acceptance by final state has an extra component $F \subseteq \mathsf{P}$ that is the subset of accepting states: in which case, $\mathsf{L}(p\alpha)$ is the set of words $\{w : p\alpha \xrightarrow{w} q\beta \text{ and } q \in F\}$. For any determinisitic context-free language L, there is a stable configuration of a DPDA with empty stack acceptance that accepts the language $\{w\$: w \in L\}$ where \$ is a new alphabet symbol, an end marker.

The DPDA problem is whether $\mathsf{L}(p\alpha) = \mathsf{L}(q\beta)$. Clearly, it is sufficient to restrict the problem to stable configurations. Moreover, one can assume that the DPDA is in normal form: if $pS \xrightarrow{a} q\alpha \in \mathsf{T}$, then $|\alpha| \leq 2$; if $pS \xrightarrow{\varepsilon} q\alpha \in \mathsf{T}$ then $\alpha = \varepsilon$; and there are no redundant transitions in T (a transition $pS \xrightarrow{a} q\alpha$ is redundant if $\mathsf{L}(q\alpha) = \emptyset$).

Example 1. Let $\mathsf{P} = \{p, r\}$, $\mathsf{S} = \{X, Y\}$ and $\mathsf{A} = \{a, b, c\}$. The basic transitions T are: $pX \xrightarrow{a} pX$, $pX \xrightarrow{b} p\epsilon$, $pX \xrightarrow{c} pX$, $rX \xrightarrow{\epsilon} p\epsilon$, $pY \xrightarrow{a} p\epsilon$, $pY \xrightarrow{b} r\epsilon$, $pY \xrightarrow{c} pYY$ and $rY \xrightarrow{\epsilon} r\epsilon$. This example of a DPDA is in normal form. □

$\mathsf{G}(p\alpha)$ is the possibly infinite state determinisitic transition graph generated by a stable configuration $p\alpha$ that abstracts from the basic ϵ-transitions. Transitions in $\mathsf{G}(p\alpha)$ are only labelled by elements of A and only relate stable configurations: $q\beta \xrightarrow{a} r\delta$ is a transition if both configurations are stable and $q\beta \xrightarrow{a} q'\beta' \xrightarrow{\varepsilon} r\delta$. The graph $\mathsf{G}(pYX)$ is pictured in Figure 1, where pYX is a configuration from Example 1. There is a path $p\alpha \xrightarrow{a_1} p_1\alpha_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n\varepsilon$

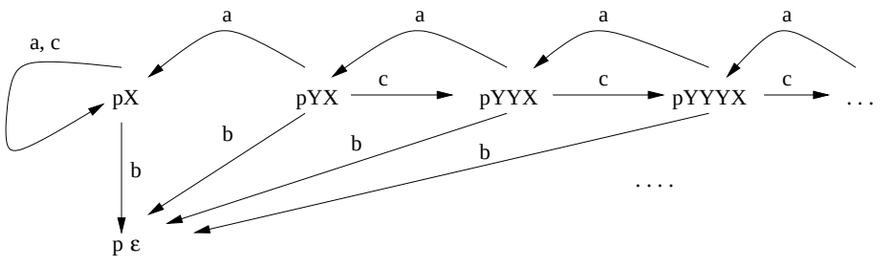


Fig. 1. The graph $\mathsf{G}(pYX)$

in $\mathsf{G}(p\alpha)$ if, and only if, $a_1 \ldots a_n \in \mathsf{L}(p\alpha)$. Because these graphs are deterministic and there are no redundant transitions, language equivalence coincides with bisimulation equivalence.

There is not an obvious relation between the lengths of stacks of equivalent configurations[1]: in the case of Example 1, $\mathsf{L}(pY^nX) = \mathsf{L}(pY^mX)$ for every m and n. Techniques for proving decidability of bisimulation equivalence, as developed

[1] A main attack on the decision problem in the 1970s examined differences between stack lengths and potentially equivalent configurations that eventually resulted in a proof of decidability for real-time DPDAs, that have no ε-transitions, [10].

in the 1990s [1], use decomposition and congruence that allows substitutivity of subexpressions in configurations. However, $\mathsf{L}(pY^n) = \mathsf{L}(pY^m)$ only if $n = m$. Moreover, the operation of stack extension is not a congruence. An important step is to provide a syntactic representation of stable DPDA configurations that dispenses with ε-transitions and that supports congruence, a process calculus that can directly generate transition graphs such as $\mathsf{G}(pYX)$. The key is *nondeterministic* pushdown automata with a single state and without ϵ-transitions, introduced by Harrison and Havel [6] as grammars, and further studied in [5,7].

Because the state is redundant, a configuration of a pushdown automaton with a single state is a sequence of stack symbols. Ingredients of such an automaton without ε-transitions, an SDA, are finite sets of stack symbols S, alphabet A and basic transitions T. Each basic transition has the form $S \xrightarrow{a} \alpha$ where $a \in \mathsf{A}$, S is a stack symbol and α is a sequence of stack symbols. A configuration of an SDA is a sequence of stack symbols whose transitions are determined by the prefix rule, assuming $\beta \in \mathsf{S}^*$: if $S \xrightarrow{a} \alpha \in \mathsf{T}$, then $S\beta \xrightarrow{a} \alpha\beta$. The language $\mathsf{L}(\alpha)$ accepted, or generated, by a configuration α is the set $\{w \in \mathsf{A}^* : \alpha \xrightarrow{w} \epsilon\}$, so acceptance is again by empty stack. Unlike pushdown automata with multiple states, language (and bisimulation) equivalence is a congruence with respect to stacking: if $\mathsf{L}(\alpha) = \mathsf{L}(\beta)$, then $\mathsf{L}(\alpha\delta) = \mathsf{L}(\beta\delta)$. An SDA can be transformed into normal form: if $S \xrightarrow{a} \alpha \in \mathsf{T}$, then $|\alpha| \leq 2$ and $\mathsf{L}(\alpha) \neq \emptyset$.

Any context-free language that does not contain the empty word ε is generable by an SDA, and so the language equivalence problem is undecidable. However, if the SDA is deterministic, then the decision problem is decidable. A deterministic SDA, more commonly known as a "simple grammar", has restricted basic transitions: if $S \xrightarrow{a} \alpha \in \mathsf{T}$ and $S \xrightarrow{a} \beta \in \mathsf{T}$, then $\alpha = \beta$. Decidability of language equivalence between configurations of an SDA was proved by Korenjak and Hopcroft in 1966 [9]. However, the languages generable by deterministic SDA are strictly contained in the languages generable by DPDA: for instance, $\{a^n b^{n+1} : n > 0\} \cup \{a^n c : n > 0\}$ is not generable by a deterministic SDA.

Instead of assuming determinism, Harrison and Havel included an extra component, $\equiv$, an equivalence relation on the stack symbols S, in the definition of an SDA that partitions S into disjoint subsets.

Example 2. The following SDA has alphabet $\mathsf{A} = \{a, b\}$ and stack symbols $\mathsf{S} = \{A, C, X, Y\}$. The partition of S is $\{\{A\}, \{C\}, \{X\}, \{Y\}\}$. The basic transitions T are $X \xrightarrow{a} YX$, $X \xrightarrow{b} \epsilon$, $Y \xrightarrow{b} X$, $A \xrightarrow{a} C$, $A \xrightarrow{b} \epsilon$ and $C \xrightarrow{b} AA$. This SDA is deterministic. □

Example 3. The set $\mathsf{S} = \{X, Y, Z\}$, $\mathsf{A} = \{a, b, c\}$ and T contains $X \xrightarrow{a} X$, $X \xrightarrow{b} \epsilon$, $X \xrightarrow{c} X$, $Y \xrightarrow{a} \epsilon$, $Y \xrightarrow{c} YY$, $Z \xrightarrow{b} \epsilon$, $Z \xrightarrow{c} Z$ and $Z \xrightarrow{c} YZ$. The partition of S is $\{\{X\}, \{Y, Z\}\}$ which means that, for example, $Y \equiv Z$. The graphs of X and Z are illustrated in Figure 2. In particular, $\mathsf{G}(Z)$ is nondeterministic because there are two c-transitions from Z. □

The relation, $\equiv$, on S is extended to an equivalence relation between sequences of stack symbols, and the same relation, $\equiv$, is used for the extension:

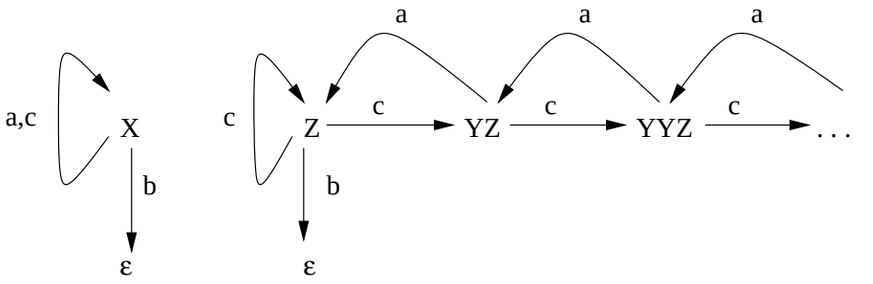


Fig. 2. The graph $\mathsf{G}(X)$ and $\mathsf{G}(Z)$

$\alpha \equiv \beta$ if, either $\alpha = \beta$, or $\alpha = \delta X \alpha'$ and $\beta = \delta Y \beta'$ and $X \equiv Y$ and $X \neq Y$. An instance of equivalent sequences, from Example 2, above, is $XXYY \equiv XXZ$ because $Y \equiv Z$. Some simple properties of $\equiv$ are: $\alpha\beta \equiv \alpha$ if, and only if, $\beta = \epsilon$; $\alpha \equiv \beta$ if, and only if, $\delta\alpha \equiv \delta\beta$; if $\alpha \equiv \beta$ and $\gamma \equiv \delta$, then $\alpha\gamma \equiv \beta\delta$; if $\alpha \equiv \beta$ and $\alpha \neq \beta$, then $\alpha\gamma \equiv \beta\delta$; if $\alpha\gamma \equiv \beta\delta$ and $|\alpha| = |\beta|$, then $\alpha \equiv \beta$.

Definition 1. The relation $\equiv$ on S is strict when the following two conditions hold. (1) If $X \equiv Y$ and $X \stackrel{a}{\longrightarrow} \alpha$ and $Y \stackrel{a}{\longrightarrow} \beta$, then $\alpha \equiv \beta$. (2) If $X \equiv Y$ and $X \stackrel{a}{\longrightarrow} \alpha$ and $Y \stackrel{a}{\longrightarrow} \alpha$, then $X = Y$.

An SDA with partition $\equiv$ is strict deterministic (or, just strict) if the relation $\equiv$ on S is strict[2]. Examples 1 and 2, above, are strict. In the case of Example 1, each partition is a singleton set and hence by (1) of Definition 1 this implies determinism. In this extreme case, it follows that $\alpha \equiv \beta$ if, and only if, $\alpha = \beta$ and, therefore, an SDA is then a simple grammar. If the partition involves larger sets, as is the case in Example 2, then constrained nondeterminism is allowed. There are transitions $Z \stackrel{c}{\longrightarrow} Z$ and $Z \stackrel{c}{\longrightarrow} YZ$. However, $Z \equiv YZ$ because $Y \equiv Z$.

Proposition 1. (1) *If* $\alpha \stackrel{w}{\longrightarrow} \alpha'$ *and* $\beta \stackrel{w}{\longrightarrow} \beta'$ *and* $\alpha \equiv \beta$ *then* $\alpha' \equiv \beta'$. (2) *If* $\alpha \stackrel{w}{\longrightarrow} \alpha'$ *and* $\beta \stackrel{w}{\longrightarrow} \alpha'$ *and* $\alpha \equiv \beta$ *then* $\alpha = \beta$. (3) *If* $\alpha \equiv \beta$ *and* $w \in \mathsf{L}(\alpha)$, *then for all words* v, *and* $a \in \mathsf{A}$, $wav \notin \mathsf{L}(\beta)$. (4) *If* $\alpha \equiv \beta$ *and* $\alpha \neq \beta$, *then* $\mathsf{L}(\alpha) \cap \mathsf{L}(\beta) = \emptyset$.

The definition of a configuration of an SDA is extended to sets of sequences of stack symbols, $\{\alpha_1, \ldots, \alpha_n\}$, written in sum form $\alpha_1 + \ldots + \alpha_n$. Two sum configurations are equal, written using $=$, if they are the same set. A degenerate case is the empty sum, written $\emptyset$. The language of a sum configuration is defined using union: $\mathsf{L}(\alpha_1 + \ldots + \alpha_n) = \bigcup \{\mathsf{L}(\alpha_i) : 1 \leq i \leq n\}$.

Definition 2. A sum configuration $\beta_1 + \ldots + \beta_n$ is *admissible*, if $\beta_i \equiv \beta_j$ for each pair of components, and $\beta_i \neq \beta_j$ when $i \neq j$.

[2] More generally, an SDA without a partition is strict deterministic if there exists a strict partition of its stack symbols. Harrison and Havel show that it is decidable whether an SDA is strict deterministic [6].

The empty sum, $\emptyset$, is therefore admissible. In [7] admissible configurations are called "associates". Some admissible configurations of Example 2, above, are XX, $ZZZ + ZZY$, $YX + Z$, $Z + YZ$ and $Z + YZ + YYZ$. An example of a configuration that is not admissible is $X + Y$ because $X \not\equiv Y$. A simple corollary of Proposition 1 is that admissibility is preserved by word transitions: if $\{\beta_1, \ldots, \beta_n\}$ is admissible, then $\{\beta' : \beta_i \xrightarrow{w} \beta',\ 1 \leq i \leq n\}$ is admissible.

A strict SDA can be determinised, by determinising the basic transitions T to T^d: for each stack symbol X and $a \in \mathsf{A}$, the transitions $X \xrightarrow{a} \alpha_1$, ..., $X \xrightarrow{a} \alpha_n$ in T are replaced by the single transition $X \xrightarrow{a} \alpha_1 + \ldots + \alpha_n$ in T^d. The sum configuration $\alpha_1 + \ldots + \alpha_n$ is admissible. Therefore, for each stack symbol X and $a \in \mathsf{A}$ there is a unique transition $X \xrightarrow{a} \sum \alpha_i \in \mathsf{T}^\mathsf{d}$. The prefix rule for generating transitions is also extended to admissible configurations: if $X_1\beta_1 + \ldots + X_m\beta_m$ is admissible and $X_i \xrightarrow{a} \sum \alpha_{ij} \in \mathsf{T}^\mathsf{d}$ for each i, then $X_1\beta_1 + \ldots + X_m\beta_m \xrightarrow{a} \sum \alpha_{1j}\beta_1 + \ldots + \sum \alpha_{mj}\beta_m$. The resulting configuration is admissible. Given a determinised strict SDA and an admissible configuration E, the graph $\mathsf{G}^\mathsf{d}(E)$ is the transition graph generated by E, except that redundant transitions, $E' \xrightarrow{a} \emptyset$, are omitted.

Example 4. In the case of Example 3, above, T^d contains: $X \xrightarrow{a} X$, $Y \xrightarrow{a} \epsilon$, $Z \xrightarrow{a} \emptyset$, $X \xrightarrow{b} \epsilon$, $Y \xrightarrow{b} \emptyset$, $Z \xrightarrow{b} \epsilon$, $X \xrightarrow{c} X$, $Y \xrightarrow{c} YY$ and $Z \xrightarrow{c} YZ + Z$. The graph $\mathsf{G}^\mathsf{d}(YX + Z)$ is pictured in Figure 3. □

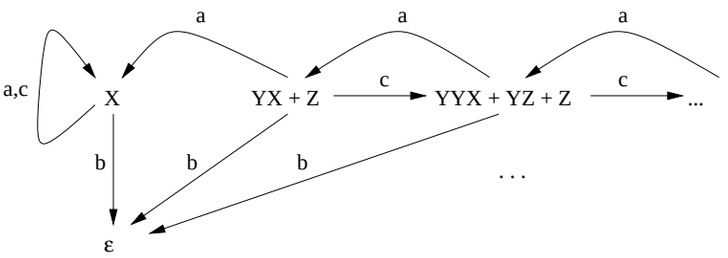


Fig. 3. The graph $\mathsf{G}^\mathsf{d}(YX + Z)$

Admissible configurations of strict SDAs generate exactly the same languages as configurations of DPDA with empty stack acceptance [6]. There is a straightforward transformation of a DPDA into a strict SDA, and DPDA configurations into admissible SDA configurations that preserve language equivalence. Assume a DPDA in normal form with sets P, S, A and T. An SDA is constructed, in stages. A) For $p, q \in \mathsf{P}$ and $X \in \mathsf{S}$, introduce an SDA stack symbol $[pXq]$. B) For transitions, the initial step is to define the following for $a \in \mathsf{A}$: if $pX \xrightarrow{a} q\epsilon \in \mathsf{T}$, then $[pXq] \xrightarrow{a} \epsilon$; if $pX \xrightarrow{a} qY \in \mathsf{T}$, then $[pXr] \xrightarrow{a} [qYr]$ for each $r \in \mathsf{P}$; if $pX \xrightarrow{a} qYZ \in \mathsf{T}$, then $[pXr] \xrightarrow{a} [qYp'][p'Zr]$ for each r and p' in P. C) $[pSq]$ is an ε-symbol, if $pX \xrightarrow{\epsilon} q\epsilon \in \mathsf{T}$. All ε-symbols are erased from the right hand side of any transition given in B. D) Finally, the

SDA is normalised. Clearly, the relation $[pSq] \equiv [pSr]$ is strict. Although the transformation does not preserve determinism, this is overcome by determinising the SDA. Any configuration $pX_1X_2 \ldots X_n$ of the DPDA is transformed into $\mathrm{sum}(p\alpha) = \sum [pX_1p_1][p_1X_2p_2] \ldots [p_{n-1}X_np_n]$ where the summation is over all $p_i \in \mathsf{P}$, after all ϵ-symbols are erased, and all components involving redundant stack symbols are removed. And $\mathsf{L}(p\alpha) = \mathsf{L}(\mathrm{sum}(p\alpha))$. An example is the conversion of the DPDA of Example 1. The resulting strict SDA is Example 4, above, when $X = [pXp]$, $Y = [pYp]$ and $Z = [pYr]$. The configuration $pYYX$ transforms to $[pYp][pYp][pXp] + [pYp][pYr] + [pYr]$. Harrison and Havel also prove the converse, that any strict SDA can be transformed into a DPDA [6]. The DPDA problem is equivalent to language equivalence between admissible configurations of a determinised strict SDA (which is the same as the bisimulation equivalence problem).

3 Heads and Tails

Assume a fixed determinised strict SDA in normal form with ingredients S, A, $\mathsf{T_d}$ and $\equiv$. We assume a total ordering on A, and we say that word u is shorter than v if $|u| < |v|$ or $|u| = |v|$ and u is lexicographically smaller than v. Let $E, F, G, \ldots$ range over admissible configurations, and $E = F$ if they are the same set of sequences. A useful notation is "the configuration E after the word u", written $E \cdot u$, that is the unique admissible configuration F such that $E \stackrel{u}{\longrightarrow} F$, which can be $\emptyset$. The language accepted by configuration E, $\mathsf{L}(E)$, is $\{u : (E \cdot u) = \epsilon\}$. Two configurations E and F are language equivalent, written $E \sim F$, if they accept the same language, $\mathsf{L}(E) = \mathsf{L}(F)$. Language equivalence can also be approximated. If $n \geq 0$, then E and F are n-equivalent, written $E \sim_n F$, provided that they accept the same words whose length is at most n: for all words w such that $|w| \leq n$, $(E \cdot w) = \varepsilon$ if, and only if, $(F \cdot w) = \varepsilon$.

Proposition 1. (1) *$E \sim F$ if, and only if, for all $n \geq 0$, $E \sim_n F$.* (2) *If $E \not\sim F$, then there is an $n \geq 0$ such that $E \sim_n F$ and $E \not\sim_{n+1} F$.* (3) *$E \sim F$ if, and only if, for all $u \in \mathsf{A}^*$, $(E \cdot u) \sim (F \cdot u)$.* (4) *$E \sim_n F$ if, and only if, for all $u \in \mathsf{A}^*$ where $|u| \leq n$, $(E \cdot u) \sim_{n-|u|} (F \cdot u)$.* (5) *If $E \sim_n F$ and $0 \leq m < n$, then $E \sim_m F$.* (6) *If $E \sim_n F$ and $F \not\sim_n G$, then $E \not\sim_n G$.*

Definition 1. For each stack symbol X, the word $w(X)$ is the shortest word in the set $\{u : (X \cdot u) = \epsilon\}$.

A feature of the decision procedure is repeating patterns within admissible configurations. An admissible configuration is written in sum form $\beta_1 + \ldots + \beta_n$ where each β_i is distinct. The operation $+$ can be extended: if E and F are admissible and $E \cup F$ is admissible and E, F are disjoint, $E \cap F = \emptyset$, then $E + F$ is the admissible configuration $E \cup F$. The operation $+$ is partial. Sequential composition, written as juxtaposition, is also used: if E and F are admissible, then EF is the configuration $\{\beta\gamma : \beta \in E \text{ and } \gamma \in F\}$, that is admissible. Some properties are: if $E + F$ is admissible and $u \in \mathsf{L}(E)$, then $uv \notin \mathsf{L}(F)$; if $E + F$ is admissible, then $\mathsf{L}(E) \cap \mathsf{L}(F) = \emptyset$; $\mathsf{L}(EF) = \{uv : u \in \mathsf{L}(E) \text{ and } v \in \mathsf{L}(F)\}$. Also,

the following identities hold: $E + \emptyset = E = \emptyset + E$, $E\emptyset = \emptyset = \emptyset E$, $E\varepsilon = E = \varepsilon E$, $(E + F)G = EG + FG$ and $G(E + F) = GE + GF$. Admissible configurations can have different "shapes", using $+$ and sequential composition.

Definition 2. $E = E_1G_1 + \ldots + E_nG_n$ is in head/tail form, if the head $E_1 + \ldots + E_n$ is admissible and at least one $E_i \neq \emptyset$, and each tail $G_i \neq \emptyset$.

Example 1. $E = YYYX + YYZ + YZ + Z$ is an admissible configuration of Example 4 of the previous section. The partition of the stack symbols is $\{\{X\}, \{Y, Z\}\}$. E has head/tail form, $YG_1 + ZG_2$, where $G_1 = YYX + YZ + Z$ and $G_2 = \varepsilon$. Also, E has head/tail form, $YYH_1 + YZH_2 + ZH_3$, where $H_1 = YX + Z$ and $H_2 = H_3 = \varepsilon$. $(E \cdot c)$ is $(YY \cdot c)H_1 + (YZ \cdot c)H_2 + (Z \cdot c)H_3 = YYYH_1 + YYZH_2 + (YZ + Z)H_3$. E cannot be presented as $YYG'_1 + YYG'_2 + YG'_3 + ZG'_4$: this is not a valid head/tail form because the head $YY + YY + Y + Z$ is not admissible ($YY \not\equiv Y$) and it is not disjoint ($YY + YY$ is not a proper sum). □

In the following, if a configuration E is presented as $E_1G_1 + \ldots + E_nG_n$, then assume that it fulfills the conditions of Definition 2 of a head/tail form. The following result lists some properties of head/tail forms. Language equivalence and its approximants are congruences with respect to $+$ and sum. Consequently, head/tail forms allow substitutivity of equivalent subexpressions into tails (because admissibility is preserved).

Proposition 2. *Assume $E = E_1G_1 + \ldots + E_nG_n$. (1) If $(E_i \cdot u) = \epsilon$, then for all $j \neq i$, $(E_j \cdot u) = \emptyset$ and $(E \cdot u) = G_i$. (2) If $(E_i \cdot u) \neq \emptyset$, then $(E \cdot u) = (E_1 \cdot u)G_1 + \ldots + (E_n \cdot u)G_n$. (3) If $H_i \neq \emptyset$, $1 \leq i \leq n$, then $E_1H_1 + \ldots + E_nH_n$ is a head/tail form. (4) If each $H_i \neq \emptyset$ and each $E_i \neq \varepsilon$ and for each j such that $E_j \neq \emptyset$ $H_j \sim_m G_j$, then $E \sim_{m+1} E_1H_1 + \ldots + E_nH_n$. (5) If $H_i \sim G_i$, $1 \leq i \leq n$, then $E \sim E_1H_1 + \ldots + E_nH_n$.*

Two configurations may have the same heads and differ in their tails, or may have the same tails and differ in their heads. If E has the head/tail form $E_1G_1 + \ldots + E_nG_n$ and F has a similar head/tail form $F_1G_1 + \ldots + F_nG_n$ involving the same tails[3], then the imbalance between E and F, relative to this presentation, is $\max\{\, |E_i|, |F_i| \;\; : 1 \leq i \leq n\}$. If the imbalance is 0, then they are the same configurations.

Definition 3. If $E = E_1G_1 + \ldots + E_nG_n$ and $F = F_1H_1 + \ldots + F_mH_m$, then F in its head/tail form is a tail extension of E in its head/tail form provided that each $H_i = K_1^iG_1 + \ldots + K_n^iG_n$, $1 \leq i \leq m$. When F is a tail extension of E, the associated extension e is the m-tuple $(K_1^1 + \ldots + K_n^1, \ldots, K_1^m + \ldots + K_n^m)$ without the G_is, and F is said to extend E by e.

Extensions are matrices. If E'' extends E' by e and E' extends E by f, then E'' extends E by ef ("matrix multiplication"). A special instance of an extension occurs when the tails are the same. If $E = E_1G_1 + \ldots + E_nG_n$ and

[3] Any pair of configurations E and F have a head/tail form involving the same tails: $E = EG$ and $F = FG$ when $G = \varepsilon$.

$F = F_1G_1 + \ldots + F_nG_n$, then F extends E by $e = (\varepsilon + \emptyset + \ldots + \emptyset, \ldots, \emptyset + \emptyset + \ldots + \varepsilon)$. The extension e is abbreviated to the identity (ε).

Example 2. The following uses Example 4 of the previous section. $E = YG_1 + ZG_2$ where $G_1 = X$ and $G_2 = \varepsilon$. $E' = YG'_1 + ZG'_2$ where $G'_1 = YX + Z$ and $G'_2 = \varepsilon$. $E'' = YG''_1 + ZG''_2$ where $G''_1 = YYX + YZ + Z$ and $G''_2 = \varepsilon$. E' extends E by $e = (Y + Z, \emptyset + \varepsilon)$ and E'' extends E' by $f = e = (Y + Z, \emptyset + \varepsilon)$. Therefore, E'' extends E by $ef = (YY + (YZ + Z), \emptyset + \varepsilon)$. □

4 The Decision Procedure

The procedure for deciding $E \sim F$ is to build a goal directed proof tree, a tableau, with initial goal $E \doteq F$, "is $E \sim F$?", using proof rules that reduce goals to subgoals. There are just three rules, presented in Figure 4. UNF, for "unfold", reduces a goal $E \doteq F$ to subgoals $(E \cdot a) \doteq (F \cdot a)$ for each a. It is complete and sound. If the goal is true, then so are all the subgoals. Soundness is the converse. A finer version, (2) of Fact 1, uses approximants: if the goal fails at level $m + 1$, then at least one subgoal fails at level m.

UNF

$$\frac{E \doteq F}{(E \cdot a_1) \doteq (F \cdot a_1) \quad \ldots \quad (E \cdot a_k) \doteq (F \cdot a_k)} \; \mathsf{A} = \{a_1, \ldots, a_k\}$$

BAL(R)

$$\frac{\begin{array}{l} F \doteq X_1H_1 + \ldots + X_kH_k \\ \quad \vdots \qquad\qquad\qquad\

Fact 1. (1) *If* $E \sim F$ *and* $a \in \mathsf{A}$*, then* $(E \cdot a) \sim (F \cdot a)$. (2) *If* $E \not\sim_{m+1} F$*, then for some* $a \in \mathsf{A}$*,* $(E \cdot a) \not\sim_m (F \cdot a)$.

Example 1. Below is an application of UNF where X, Y and Z are from Example 4 of Section 2.

$$\frac{YX + Z \doteq YYX + YZ + Z}{X \doteq YX + Z \quad \epsilon \doteq \epsilon \quad YYX + YZ + Z \doteq YYYX + YYZ + YZ + Z}$$

The three subgoals are the result after a, b and c. □

If $E' \doteq F'$ is a subgoal that is a result of m consecutive applications of UNF (and no other rule) to $E \doteq F$, then there is an associated word u such that $|u| = m$ and $E' = (E \cdot u)$ and $F' = (F \cdot u)$. The other two rules in Figure 4 are conditional that involve two premises: the second premise goal reduces to the subgoal beneath it provided that the first premise is above it (on the path back to the root goal). They are BAL rules, for "balance", involving substitution of subexpressions, that allow a goal to be reduced to a balanced subgoal where the imbalance between the configurations is bounded.

Example 2. An application of BAL(L) uses stack elements of Example 2 of Section 2.

$$\dfrac{\dfrac{XXXXXX \doteq AAAAAA}{YXXXXXX \doteq CAAAAA \quad \cdots}\,\text{UNF}}{YXAAAAA \doteq CAAAAA}\,\text{BAL(L)}$$

The second goal is the result of UNF when the label is a (and the other subgoal for b is omitted). $w(X) = b$, so $m = 1$. Therefore, BAL(L) applies to the second goal: $X_1 = X$, $H_1 = XXXXX$, $E_1 = YX$ and $F = AAAAAA$. So H_1 is replaced with $(F \cdot b) = AAAAA$. The imbalance between configurations of the last goal is 2. □

An application of BAL is said to use F, if F is the configuration in the initial goal of the rule, see Figure 4. The BAL rules are sound and complete. Completeness is straightforward. Soundness is more intricate. First, "global" soundness of the proof system is explained. If there is a successful tableau whose root is false, then there is a branch of the tableau within which each subgoal is false. The idea is refined using approximants. If the root is false then there is an offending branch (of false goals) in the tableau within which the approximant indices decrease whenever rule UNF has been applied. Soundness of an application of BAL is that if the two premise goals belong to an offending branch, then the subgoal preserves the level of falsity of the second premise goal.

Proposition 1. (1) *If* $X_1H_1 + \ldots + X_kH_k \sim F$ *and* $E_1H_1 + \ldots + E_kH_k \sim F'$*, then* $E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k)) \sim F'$. (2) *If* $X_1H_1 + \ldots + X_kH_k \sim_{n+m} F$ *and* $E_1H_1 + \ldots + E_kH_k \not\sim_{n+1} F'$ *and each* $E_i \neq \varepsilon$ *and* $m \geq \max\{|w(X_i)| : E_i \neq \emptyset\}$*, then* $E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k)) \not\sim_{n+1} F'$.

In Example 2, above, BAL(L) is applied after UNF. However, the other two rules UNF and BAL(R) also apply. It is intended that there be a unique tableau associated with any initial goal. So restrictions will be placed on which rule is to be applied when. First, the initial premise of a BAL is the one that is "closest" to the goal and, therefore, the one that involves the least number of applications of UNF. To resolve which rule should be applied, the following priority order is assumed: (1) if BAL(L) is permitted, then apply BAL(L), (2) if BAL(R) is permitted, then apply BAL(R), (3) otherwise, apply UNF. However, whether an application of BAL is permitted involves more than fulfillment of the side condition. It also depends on the previous application of a BAL.

Initially, either BAL is permitted provided that its side condition is true. If an application of BAL uses F, then the resulting goal contains the configuration $E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k))$. E_i is a "top" of the application of BAL and $(F \cdot w(X_i))$ is a "bottom". Assume an application of BAL(L). A subsequent application of BAL(L) is permitted provided the side condition of the rule is fulfilled. However, BAL(R) is not permitted until a bottom of the previous application of BAL(L) is exposed and the side condition of the rule is true. Between the application of BAL(L) and the goal $G_1 \doteq H_1$, below,

$$\begin{array}{rl} & F \\ & \vdots \quad \text{BAL(L)} \\ E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k)) \doteq & H \\ \vdots \quad \vdots & \quad \text{UNFs} \\ (F \cdot w(X_i)) = G_1 \doteq & H_1 \\ \vdots \quad \vdots & \\ G_k \doteq & H_k \end{array}$$

there are no other applications of BAL(L), and G_1 is a bottom, $(F \cdot w(X_i))$, of the previous application of BAL(L). BAL(R) is now permitted provided it uses configuration G_i, $i \geq 1$, and the side condition holds. BAL(R) is not permitted using a configuration from a goal above $G_1 \doteq H_1$, even when the side condition is true. The strategy is to apply a BAL rule whenever it is permitted, and if both BAL rules are permitted, then priority lies with BAL(L). If BAL(R) is applied, then the strategy is to repeatedly apply BAL(R), and to use UNF otherwise. BAL(L) is only permitted once a bottom of the previous application of BAL(R) becomes the right hand configuration of a goal and the side condition holds. The consequence is that when building a tableau proof tree, there is just one choice of which rule to apply next to any subgoal.

Example 3. An initial part of the tableau, continuing on from Example 2 above, is in Figure 5. At goal $(*)$, BAL(L) is applied. Either of the premises (1) and (2) could be the initial premise for the application: however, by the discussion above it is the lower premise (2). □

$$\dfrac{\dfrac{\dfrac{\dfrac{XX^5 \doteq AA^5}{YXX^5 \doteq CA^5}\,\text{UNF} \qquad X^5 \doteq A^5}{(1)\ YXA^5 \doteq CA^5}\,\text{BAL(L)}}{\emptyset \doteq \emptyset \qquad (2)\ XXA^5 \doteq AA^6}\,\text{UNF}}{\dfrac{(*)\ YXXA^5 \doteq CA^6 \qquad XA^5 \doteq A^6}{YXA^6 \doteq CA^6}\,\text{BAL(L)}}\,\text{UNF}$$

Fig. 5. Part of a tableau

Example 4. Below is the initial part of the tableau for Example 1, above.

$$\dfrac{(\mathbf{1}) \quad \epsilon \doteq \epsilon \qquad \dfrac{(*)\ YX + Z \doteq YYX + YZ + Z}{YYX + YZ + Z \doteq YYYX + YYZ + YZ + Z}\,\text{UNF}}{YYYX + YYZ + YZ + Z \doteq YYYX + YYZ + YZ + Z}\,\text{BAL(L)}$$

where (**1**) is the subtableau

$$\dfrac{\dfrac{\dfrac{(**)\ X \doteq YX + Z}{X \doteq X \quad \epsilon \doteq \epsilon \qquad X \doteq YYX + YZ + Z}\,\text{UNF}}{X \doteq YYX + YZ + Z}\,\text{BAL(R)}}{X \doteq YX + Z \quad \epsilon \doteq \epsilon \qquad \dfrac{X \doteq YYYX + YYZ + YZ + Z}{X \doteq YYX + YZ + Z}\,\text{BAL(R)}}\,\text{UNF}$$

The premise $(*)$ is the initial premise for the application of BAL(L), and $(**)$ is the initial premise for the first BAL(R). The leaf goals are either identities or repeats. In fact, it will turn out that this partial tableau is the completed successful tableau that establishes that $\mathsf{L}(pYX) = \mathsf{L}(pYYX)$ of Example 1 of Section 2. □

For the tableau construction to be a decision procedure, a notion of final goal is needed so that a tableau can be terminated. The tableau proof rules are locally complete, if a goal is true then so are subgoals. Consequently, if an obviously false subgoal is reached, then the root goal is also false. So the criterion for being an unsuccessful final goal is that it is obviously false. This occurs when the goal has the form $\emptyset \doteq E$ or $E \doteq \emptyset$ and $E \neq \emptyset$. The tableau proof rules are also locally sound, if all the subgoals are true then so is the goal. Therefore, if an obviously true subgoal, $E \doteq E$, is reached then it should count as a successful final leaf. However, the tableau proof rules are sound in a finer version. In the case of UNF, if the goal is false at level $m+1$, $E \not\sim_{m+1} F$, then at least one subgoal fails at level m, $(E \cdot a) \not\sim_m (F \cdot a)$. And applications of BAL preserve the falsity

index. Consequently, if a subgoal $E \doteq F$ is repeated in a branch, and there is at least one application of UNF between them, then the second occurrence of $E \doteq F$ can also count as a successful final goal. If the root of the tableau is false, then there is an offending path of false goals in the tableau within which the approximant indices decrease whenever UNF is applied. Consider the branch with $E \doteq F$ occurring twice: if this were an offending branch, then at the first occurrence, there is a least $n \geq 0$ such that $E \sim_n F$ and $E \not\sim_{n+1} F$. Therefore, at the second occurrence $E \not\sim_{(n+1)-k} F$ where k is the number of applications of UNF between the two; this is a contradiction when $k \geq 1$.

A repeat is an instance of a more general situation where goals may be growing in size, formally captured below by the "extension theorem". Roughly speaking, in a branch if there are goals where the rates of change of tails are repeating, then there is a successful final goal. A repeat is an instance when the rate of change is zero. In a long enough branch with multiple applications of BAL, there must be goals within the branch that have the same heads. The idea is to discern patterns of relations between their tails. Definition 3 of the previous section is lifted to goals. Assume $E = E_1H_1 + \ldots + E_nH_n$, $F = F_1H_1 + \ldots + F_nH_n$, $E' = E'_1G_1 + \ldots + E'_mG_m$ and $F' = F'_1G_1 + \ldots + F'_mG_m$ and goal h is $E \doteq F$ and goal g is $E' \doteq F'$. Goal h extends g by extension e, if E extends E' by e. A goal $E \doteq F$ is true at level m if $E \sim_m F$.

Theorem 1. [The extension theorem] *Assume there are two families of goals $g(i)$, $h(i)$, $1 \leq i \leq 2^n$, and each goal $g(i)$ has the form $E_1G^i_1 + \ldots + E_nG^i_n \doteq F_1G^i_1 + \ldots + F_nG^i_n$ and each goal $h(i)$ has the form $E_1H^i_1 + \ldots + E_nH^i_n \doteq F_1H^i_1 + \ldots + F_nH^i_n$. Assume extensions $e_1, \ldots, e_n$ such that for each e_j and $i \geq 0$, $g(2^j i + 2^{j-1} + 1)$ extends $g(2^j i + 2^{j-1})$ by e_j and $h(2^j i + 2^{j-1} + 1)$ extends $h(2^j i + 2^{j-1})$ by e_j. If each goal $g(i)$ is true at level m, $i : 1 \leq i \leq 2^n$, and each goal $h(j)$, $j : 1 \leq j < 2^n$, is true at level m, then $h(2^n)$ is true at level m.*

A simple instance is explained. Consider the proof tree of Example 3. There is a branch where the goals are expanding as follows: $YXA^5 \doteq CA^5$, ..., $YXA^6 \doteq CA^6$, ..., $YXA^7 \doteq CA^7$, ... And between these goals there is at least one application of UNF. To instantiate the extension theorem $n = 1$. The families of goals are as follows. $g(1) : YXG^1 \doteq CG^1$ where $G^1 = A^5$, $g(2) = h(1) : YXG^2 \doteq CG^2$ where $G^2 = A^6$, $h(2) : YXH^2 \doteq CH^2$ where $H^2 = A^7$. The extension is (A): $g(2)$ extends $g(1)$ by (A) and $h(2)$ extends $h(1)$ by (A). The theorem provides the following result: for any m, if $YXA^5 \sim_m CA^5$ and $YXA^6 \sim_m CA^6$, then $YXA^7 \sim_m CA^7$. This justifies that the subgoal $YXA^7 \doteq CA^7$ is a successful final goal. The argument is the same as for a repeating goal, above.

Definition 1. Assume a branch of goals $d(0), \ldots, d(l)$. The goal $d(l)$ obeys the extension theorem if there are goals $g(i)$, $h(i)$, $1 \leq i \leq 2^n$ and extensions $e_1, \ldots, e(n)$ as described in Theorem 1, and the goals belong to $\{d(0), \ldots d(l)\}$, and $h(2^n)$ is $d(l)$ and there is at least one application of UNF between goal $h(2^n - 1)$ and $d(l)$.

The second occurrence of a repeating goal in a branch obeys the extension theorem if there is at least one application of UNF between the two occurrences. Assume it has the form $E_1G_1^i + \ldots + E_nG_n^i \doteq F_1G_1^i + \ldots + F_nG_n^i$. Except for $h(2^n)$, the goals $g(i)$ and $h(i)$ are the first occurrence of the repeating goal, and each extension is the identity, (ϵ).

Definition 2. Assume a branch of goals $g(0), \ldots, g(n)$ where $g(0)$ is the root goal. The goal $g(n)$ is a final goal in the following circumstances.

1. If $g(n)$ is an identity $E \doteq E$, then $g(n)$ is a successful final goal
2. If $g(n)$ obeys the extension theorem, then $g(n)$ is a successful final goal
3. If $g(n)$ has the form $E \doteq \emptyset$ or $\emptyset \doteq E$ and $E \neq \emptyset$, then $g(n)$ is an unsuccessful final goal.

Lemma 1. *In any infinite branch of goals $g(0), \ldots, g(n), \ldots$ where $g(0)$ is the root goal, there is an n such that $g(n)$ is a final goal.*

The deterministic procedure that decides whether $E \sim F$ is straightforward, and is defined iteratively.

1. Stage 0: start with the root goal $g(0)$, $E \doteq F$, that becomes a frontier node of the branch $g(0)$.
2. Stage $n+1$: if a current frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ is an unsuccessful final goal, then halt and return "unsuccessful tableau"; if each frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ is a successful final goal, then return "successful tableau"; otherwise, for each frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ that is not a final goal, apply the next rule to it, and the subgoals that result are the new frontier nodes of the extended branches.

Theorem 2. (1) *If $E \not\sim F$, then the decision procedure terminates with "unsuccessful tableau".* (2) *If $E \sim F$, then the decision procedure terminates with "successful tableau".*

Theorem 2 establishes decidability of language equivalence between DPDA configurations. Part 1 of Theorem 2 is straightforward. The other half, part 2, is more difficult and uses Lemma 1, above. However, a more refined analysis of the lemma should produce an elementary complexity upper bound. Currently, the proof of Lemma 1 abstracts from "oscillation" whereby goals can increase and decrease their sizes. A more involved proof would establish that a boundedly finite branch of goals contains a final goal.

The proof of Theorem 1, the extension theorem, follows from the following much simpler result.

Lemma 2. *If $E = E_1G_1 + \ldots + E_nG_n$, $F = F_1G_1 + \ldots + F_nG_n$, $E' = E_1H_1 + \ldots + E_nH_n$ and $F' = F_1H_1 + \ldots + F_nH_n$ and $E \sim_m F$ and $E' \not\sim_m F'$, then there is a word u, $|u| \leq m$, and an i such that either $(E' \cdot u) = H_i$ and $(F' \cdot u) = (F_1 \cdot u)H_1 + \ldots + (F_n \cdot u)H_n$ and $(E' \cdot u) \not\sim_{m-|u|} (F' \cdot u)$, or $(F' \cdot u) = H_i$ and $(E' \cdot u) = (E_1 \cdot u)H_1 + \ldots + (E_n \cdot u)H_n$ and $(E' \cdot u) \not\sim_{m-|u|} (F' \cdot u)$.*

References

1. Burkart, O., Caucal, D., Moller, F., and Steffen, B. (2001). Verification on infinite structures. In *Handbook of Process Algebra*, edited Bergstra, J., Ponse, A., and Smolka, S. 545-623, *North Holland*.
2. Christensen, S., Hirshfeld, Y. and Moller, F. (1993). Bisimulation equivalence is decidable for all basic parallel processes. *Lecture Notes in Computer Science*, **715**, 143-157.
3. Christensen, S., Hüttel, H., and Stirling, C. (1995). Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, **121**, 143-148.
4. Ginsberg, S., and Greibach, S. (1966). Deterministic context-free languages. *Information and Control*, 620-648.
5. Harrison, M. (1978). *Introduction to Formal Language Theory*, Addison-Wesley.
6. Harrison, M., and Havel, I. (1973). Strict deterministic grammars. *Journal of Computer and System Sciences*, **7**, 237-277.
7. Harrison, M., Havel, I., and Yehudai, A. (1979). On equivalence of grammars through transformation trees. *Theoretical Computer Science*, **9**, 173-205.
8. Hüttel, H., and Stirling, C. (1991). Actions speak louder than words: proving bisimilarity for context free processes. *Proceedings 6th Annual Symposium on Logic in Computer Science*, IEEE Computer Science Press, 376-386.
9. Korenjak, A and Hopcroft, J. (1966). Simple deterministic languages. *Procs. 7th Annual IEEE Symposium on Switching and Automata Theory*, 36-46.
10. Oyamaguchi, M., Honda, N., and Inagaki, Y. (1980). The equivalence problem for real-time strict deterministic languages. *Information and Control*, **45**, 90-115.
11. Sénizergues, G. (1997). The equivalence problem for deterministic pushdown automata is decidable. *Lecture Notes in Computer Science*, **1256**, 671-681.
12. Sénizergues, G. (2001). L(A) = L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, **251**, 1-166.
13. Stirling, C. (1998). Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, **195**, 113-131.
14. Stirling, C. (2001). Decidability of DPDA equivalence. *Theoretical Computer Science*, **255**, 1-31.

Semidefinite Programming Based Approximation Algorithms

Uri Zwick *

School of Computer Science
Tel Aviv University, Tel Aviv 69978, Israel
zwick@cs.tau.ac.il
http://www.cs.tau.ac.il/~zwick/

Abstract. The talk would describe the use of semidefinite programming in the development of approximation algorithms for combinatorial optimization problems. The talk would start with a definition of semidefinite programming. No prior knowledge of the subject would be assumed. It would then briefly cover Lovász's ϑ-function, the MAX CUT approximation algorithm of Goemans and Williamson, the coloring algorithm of Karger, Motwani and Sudan, the MAX 3-SAT algorithm of Karloff and Zwick, and time permitting more modern developments.

* Work supported in part by **The Israel Science Foundation** founded by The Israel Academy of Sciences and Humanities.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, p. 57, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Hard Sets and Pseudo-random Generators for Constant Depth Circuits

Manindra Agrawal

Department of Computer Science
IIT Kanpur
Kanpur 208016, India
manindra@iitk.ac.in

Abstract. It is shown that the existence of a set in E that is hard for constant depth circuits of subexponential size is equivalent to the existence of a true pseudo-random generator against constant depth circuits.

1 Introduction

Pseudo-random generators against a class of circuits are functions that take a random seed as input and output a sequence of bits that cannot be distinguished from a truly random sequence by any circuit in the class. They play an important role in many areas, particularly in cryptography and derandomization (see, e.g., [BM84,Yao82]). In this paper, we will be interested in derandomization aspect of pseudo-random generators, and therefore, will use the following definition (as given in [NW94]):

Definition 1. *For the class of circuits $\mathcal{C}$, function G is called a* $(\ell \mapsto n)$ pseudo-random generator against $\mathcal{C}$ *if*

- $G = \{G_n\}_{n>0}$ *with* $G_n : \{0,1\}^{\ell(n)} \mapsto \{0,1\}^n$,
- G_n *is computable in time* $2^{O(\ell(n))}$,
- *for every* n, *and for every circuit* $C \in \mathcal{C}$ *having* n *input bits,*

$$| \operatorname{prob}_{x\in\{0,1\}^n}\{C(x) = 1\} - \operatorname{prob}_{y\in\{0,1\}^{\ell(n)}}\{C(G_n(y)) = 1\} | \leq \frac{1}{n}.$$

To derandomize a randomized algorithm, one uses a $(\ell \mapsto n)$ pseudo-random generator against a class of circuits that include the circuit family coding the algorithm, and feed the output of the generator as random input bits to the algorithm for each value of the seed, and then calculate the fraction of ones in the output. Of course, this modified algorithm takes more time—the time taken to compute the generator for every seed value times the time to run the algorithm on every output of the generator. To minimize the time taken, one needs to reduce $\ell(n)$: the best that can be achieved is $\ell(n) = O(\log n)$ and then the increase in time complexity is by a factor of polynomial only. Pseudo-random generators that achieve this seed size are called true pseudo-random generators:

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 58–69, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Definition 2. *For the class of circuits $\mathcal{C}$, function G is called a* true pseudo-random generator against $\mathcal{C}$ *if G is a $(\ell \mapsto n)$ pseudo-random generator against $\mathcal{C}$ with $\ell(n) = O(\log n)$.*

While true pseudo-random generators against specific algorithms (i.e., the class against which the generator works include circuits for a specific algorithm only) are known, very few *unconditional* pseudo-random generators are known against natural classes of circuits. Perhaps the most notable amongst these are $((\log n)^{O(d)} \mapsto n)$ pseudo-random generators against the class of depth d and size n circuits [Nis91].

In a seminal work, Nisan and Wigderson [NW94] exhibited a connection between pseudo-random generators and *hard-to-approximate* sets in E:

Definition 3. *For a set A and circuit C with n input bits, let*

$$\mathrm{adv}_C(A) = \mid \mathrm{prob}_{x\in\{0,1\}^n}[C(x) = A(x)] - \mathrm{prob}_{x\in\{0,1\}^n}[C(x) \neq A(x)] \mid .$$

Here we identify A with its characteristic function. For a size bound $s(n)$ of circuits, let $\mathrm{adv}_{s(n)}(A)$ be the maximum of $\mathrm{adv}_C(A)$ where C varies over all size $s(n)$ circuits.

Set $A \in$ E is hard-to-approximate *by circuits of size $s(n)$ if $\mathrm{adv}_{s(n)}(A) \leq \frac{1}{s(n)}$.*

Nisan and Wigderson showed that:

Nisan-Wigderson Theorem 1. [NW94] *There exist $(\ell \mapsto s(\ell^c))$ pseudo-random generators against class of size $s(\ell^c)$ circuits (for some size bound s and constant $c > 0$) if and only if there exist sets in* E *that are hard-to-approximate by circuits of size $s(\ell^d)$ (for some constant $d > 0$).*

In fact, the pseudo-random generator of [Nis91] is constructed using the above theorem and the fact that there exists a set (e.g. PARITY [Has86]) that is hard-to-approximate by circuits of size $2^{\ell^{\frac{1}{O(d)}}}$ and depth d (the above theorem of Nisan and Wigderson holds in the presence of depth restriction too).

An interesting special case is that of true pseudo-random generators, i.e., when $s(\ell) = 2^{O(\ell)}$. In that case, [NW94] showed that both the constants c and d can be set to one, and thus we get:

Nisan-Wigderson Theorem 2. [NW94] *There exist true pseudo-random generators against class of size $2^{\delta\cdot\ell}$ circuits for some constant $0 < \delta < 1$ if and only if there exist sets in* E *that are hard-to-approximate by circuits of size $2^{\epsilon\cdot\ell}$ for some constant $0 < \epsilon < 1$.*

One of the major implication of the existence of above true pseudo-random generators is that BPP = DP. In the following, we restrict our attention to true pseudo-random generators only as these have the most interesting implications. So, $n = 2^{O(\ell)}$ throughout the paper.

Although [NW94] provides evidence that true pseudo-random generators exist, it is not clear that hard-to-approximate sets, as required, do exist in E. On the other hand, it is easier to believe that there exist sets in E that cannot be solved by subexponential size circuits—in other words, there is a set in E such that $\text{adv}_{2^{\epsilon \cdot \ell}}(A) < 1$ for some $0 < \epsilon < 1$. Therefore, a major line of research in the last ten years has been to construct true pseudo-random generators from this weaker assumption. The approach taken was to start with a set A in E with $\text{adv}_{2^{\epsilon \cdot \ell}}(A) < 1$, and derive another set $B \in \text{E}$ from A such that B is hard-to-approximate by $2^{\epsilon' \cdot \ell}$ size circuits as required in the above theorem.

The above aim was achieved in three steps. First, [BFNW93] constructed—starting from a set $A^1 \in \text{E}$ with $\text{adv}_{2^{5\epsilon\ell}}(A^1) < 1$—a set $A^2 \in \text{E}$ such that $\text{adv}_{2^{3\epsilon\ell}}(A^2) < 1 - \frac{1}{\ell^2}$. Then, in [Imp95], a third set A^3 was constructed from A^2 with $\text{adv}_{2^{2\epsilon\ell}}(A^2) < \frac{5}{6}$, and finally in [IW97] a set A^4 was constructed from A^3 with $\text{adv}_{2^{\epsilon\ell}}(A^4) \leq \frac{1}{2^{\epsilon \cdot \ell}}$ thus achieving the desired generalization of the Nisan-Wigderson Theorem 2. In [STV99] two alternative constructions were given for the same result.

The work in this paper is motivated by the following question: what is the hardness condition needed for constructing true pseudo-random generators against classes of circuits more restricted than the class of polynomial-sized circuits (the class of circuits in the Nisan-Wigderson Theorem 2 is polynomial-sized in the generator output size, and exponential-sized in the generator input size)? A natural way of defining such circuits is by restricting their depth. So we can pose this question for several natural classes of small depth circuits, e.g., AC^0, TC^0, NC^1, NC, etc. In analogy with the above result, we should perhaps expect that to construct pseudo-random generators against polynomial-sized circuits of depth d, we need a hard set against subexponential sized circuits of depth $O(d)$.

We first observe that the constructions given in [BFNW93,Imp95,IW97] have the following property: starting with a set that is hard to compute by the class of circuits of size $2^{5\epsilon\ell}$ and depth d, the constructed set is hard-to-approximate by circuits of size $2^{\epsilon \cdot \ell}$ and depth $d - O(1)$ (for some $\epsilon, \alpha > 0$) *provided the majority gate* is allowed in the original class of circuits. This implies that for all circuit classes $\mathcal{C}$ that include TC^0, one can construct true pseudo-random generators against $\mathcal{C}$ using a set in E that is hard to compute by subexponential sized circuits of the same depth (within a constant factor) as in $\mathcal{C}$.

Therefore, our question is answered for all the well-known circuit classes except for the class AC^0. AC^0 circuits are polynomial-sized constant depth circuits and it is known that they *cannot* compute the majority function [Has86]. Therefore, the construction of [BFNW93,Imp95,IW97] does not give the expected result. Further, this seems to be a fundamental bottleneck as the other two constructions given in [STV99] also require at least threshold gates. So we have a intriguing situation here: even though there exist nearly true pseudo-random generators against AC^0 circuits (given by Nisan [Nis91]) that are *unconditional*, we *do not* seem to get *conditional* true pseudo-random generators against AC^0 under a condition whose stronger forms give true pseudo-random generators against larger classes of circuits! It is useful to note here that true

pseudo-random generators against AC^0 circuits are interesting in their own right: their existence would imply that approximate DNF-counting can be derandomized [KL83].

In this paper, we close this gap in our knowledge to show that:

Theorem 1. *There exist true pseudo-random generators against class of size* $2^{\delta \cdot \ell}$ *and depth* $O(d)$ AC^0 *circuits for some constant* $0 < \delta < 1$ *if and only if there exist a set in* E *that cannot be computed by* AC^0 *circuits of size* $2^{\gamma \cdot \ell}$ *and depth* $O(d)$ *for some constant* $0 < \gamma < 1$.

The idea is to exploit the unconditional pseudo-random generators of Nisan. The generator of Nisan stretches a seed of size $(\log n)^{O(d)}$ to n bits and works against depth d, size n AC^0 circuits. Moreover, every output bit of the generator is simply a parity of a subset of seed bits. Now the crucial observation is that parity of poly(log n) bits *can* be computed by AC^0 circuits, and so if we compose the Nisan generator with any given circuit C of depth d and size n, we get another AC^0 circuit of a (slightly) larger depth and size that has only poly(log n) input bits (as opposed to n in C) and yet the circuit accepts roughly the same fraction of inputs as C. A careful observation of the constructions of [BFNW93,Imp95,NW94,NW94] yields that if the pseudo-random generator constructed through them needs to stretch a seed of ℓ bits to only poly(ℓ) bits (instead of $2^{\epsilon \cdot \ell}$ bits), then we need to start from a set in E that is hard to compute by circuits of size $2^{\delta' \cdot \ell}$, depth d that have majority gates over only poly(ℓ) bits (instead of over $2^{O(\ell)}$ bits). Such majority gates can be replaced by AC^0 circuits of size $2^{o(\ell)}$. Therefore, we only require sets in E that are hard to compute by size $2^{\delta \cdot \ell}$ and depth d' AC^0 circuits! A minor drawback of the result is that the true pseudo-random generators that we obtain approximate the fraction of inputs accepted by a circuit C within $\frac{1}{\text{poly}(\log n)}$ as opposed to $\frac{1}{n}$ in all the other cases. However, for many applications, e.g., derandomizing approximate DNF-counting, this weaker approximation is sufficient.

The organization of the paper is as follows: in the next section we analyze the existing constructions and in Section 3 we give our construction.

2 Depth Increase in Existing Constructions

The construction in [BFNW93,Imp95,IW97,NW94] can be divided into five stages:

Stage 1. Given a set A_1 in E such that $\text{adv}_{2^{\epsilon_1 \ell}}(A_1) < 1$, construct a function $f = \{f_\ell\}$ in E such that for any $\epsilon_f < \epsilon_1$, and for any circuit C of size $2^{\epsilon_f \ell}$, the fraction of inputs on which C can compute f_ℓ correctly is at most $1 - \frac{1}{\ell^2}$. This construction was given in [BFNW93].

Stage 2. From the function f construct a set $A_2 \in$ E such that for any $\epsilon_2 < \epsilon_f$, $\text{adv}_{2^{\epsilon_2 \ell}}(A_2) < 1 - \frac{1}{\ell^3}$. This construction was given in [GL89].

Stage 3. From the set A_2 construct a set $A_3 \in$ E such that for any $\epsilon_3 < \epsilon_2$, $\text{adv}_{2^{\epsilon_3 \ell}}(A_3) < \frac{15}{16}$. This construction was given in [Imp95].

Stage 4. From the set A_3 construct a set $A_4 \in \mathrm{E}$ such that for any $\epsilon_4 < \epsilon_3$, $\mathrm{adv}_{2^{\epsilon_4 \ell}}(A_4) < \frac{1}{2^{\epsilon_4 \ell}}$. This construction was given in [IW97].

Stage 5. using the set A_4, construct a true pseudo-random generator $G = \{G_n\}$ with $G_n : \{0,1\}^{O(\log n)} \mapsto \{0,1\}^n$ against circuits of size n. This, of course, was given in [NW94].

We now describe each of these constructions. The correctness of all the constructions is shown using the contrapositive argument: given a circuit family that solves the constructed set (or function) with the specified advantage, we construct a circuit family that solves the original set (or function) with an advantage that contradicts the hardness assumption about the set. For our purposes, the crucial part in these arguments would be the depth and size increase in the constructed circuit family over the given circuit family. We do not need to worry about the complexity of the constructing the new set from the original one—this is an important to keep in mind as often this complexity is very high (e.g., in Stage 1 and Stage 4).

Several times in the constructions below, we make use of the following (folklore) fact about computing parity or majority of ℓ bits:

Proposition 1. *The parity or majority of ℓ bits can be computed by* AC^0 *circuits of size $O(2^{\ell^{\frac{4}{d}}})$ and depth d.*

Hastad [Has86] provided a (fairly tight) corresponding lower bound:

Lemma 1. *The parity or majority of ℓ bits cannot be computed by* AC^0 *circuits of size $2^{\ell^{\frac{1}{d+1}}}$ and depth d.*

2.1 Stage 1: Analyzing Babai-Fortnow-Nisan-Wigderson's Construction

Construction of f Function f is an small degree, multi-variate polynomial extension of the set A_1 over a suitable finite extension field of F_2. More specifically, function $f(x)$, $|x| = \ell$, is defined as follows (we assume ℓ to be a power to two for convenience):

Fix field $F = F_{\ell^2}$. Let $k = \frac{\ell}{2 \log \ell}$. Define polynomial $P(y_1, y_2, \ldots, y_k)$ over F as:

$$P(y_1, y_2, \ldots, y_k) = \sum_{v_1 : |v_1| = \log \ell} \cdots \sum_{v_k : |v_k| = \log \ell} A_1(v_1 v_2 \cdots v_k) \cdot \prod_{i=1}^{k} \delta_{v_i}(y_i),$$

where

$$\delta_{v_i}(y_i) = \frac{\prod_{v : |v| = \log \ell \wedge v \neq v_i} (y_i - v)}{\prod_{v : |v| = \log \ell \wedge v \neq v_i} (v_i - v)}.$$

Let $x = x_1 x_2 \cdots x_k$ with $|x_i| = 2 \log \ell$. Then,

$$f(x) = P(x_1, x_2, \ldots, x_k).$$

Polynomial P has $k = \frac{\ell}{2 \log \ell}$ variables and each variable has degree at most ℓ.

Correctness of construction. Suppose that a family of circuits $\{C_\ell\}$ of size $2^{\epsilon_f \ell}$ exists such that for every ℓ, C_ℓ can correctly compute f on more than $1-\frac{1}{\ell^2}$ fraction of inputs. We use this circuit family to construct a circuit family that correctly decides f, and therefore A_1, everywhere.

Fix ℓ and x, $|x| = \ell$. String x can be viewed as a point in the k-dimensional vector space over F_{ℓ^2}. Select a random line passing through x in this space. It is easily argued that with probability at least $\frac{3}{4}$, on such a line, C_ℓ will correctly compute f on at least $1-\frac{4}{\ell^2}$ fraction of points. Notice that when restricted to such a line, polynomial P reduces to a univariate polynomial P' of degree at most $\frac{\ell^2}{2\log \ell}$. Randomly select $\frac{\ell^2}{2\log \ell}+1$ points on this line and use circuit C_ℓ to find out the value of f on these. Clearly, with probability at least $1-\frac{3}{\log \ell}$ the computed value of f would be correct on all the points. Interpolate polynomial P' using these values and then compute the value of $f(x)$ using P'. The probability that $f(x)$ is correctly computed is at least $\frac{3}{4}\cdot(1-\frac{3}{\log \ell}) > \frac{2}{3}$. Repeat the same computation with different random choices ℓ^2 times and take the value occurring maximum number of times as the value of $f(x)$. The probability that this is wrong would be less than $\frac{1}{2^\ell}$. Finally, fix a setting of random bits that work for all 2^ℓ different x's. The circuit implementing this algorithm correctly computes f everywhere (the circuit is non-uniform though).

Let us now see what is the size and depth of this circuit, say C', as compared to C_ℓ. Once all the random choices are fixed, C' just needs to use C_ℓ on $\frac{\ell^2}{2\log \ell}+1$ different inputs (computed by xoring a fixed string to x), and then take a linear combination of the output values[1]. As there are $O(\ell^2)$ outputs each of size ℓ, this can be done by a AC^0 circuit of size $2^{o(\ell)}$. Thus the size of the circuit C' is at most $2^{\epsilon_1 \ell}$ as long as $\epsilon_1 > \epsilon_f$ contradicting the assumption. Notice that depth of C' is only a constant more than of C_ℓ and C does not have any majority gate except those already present in C_ℓ.

2.2 Stage 2: Analyzing Goldreich-Levin's Construction

Construction of A_2 Set A_2 is defined as: $xr \in A_2$ iff $|x| = |r| = \ell$ and $f(x)\cdot r = 1$ where '$\cdot$' is the inner product.

Correctness of the construction. Assume that a circuit family $\{C_\ell\}$ of size $2^{\epsilon_2 \ell}$ is given such that $\mathrm{adv}_{C_\ell}(A_2) \geq 1-\frac{1}{\ell^3}$. As we later need this result for smaller advantages too, we give the construction assuming that $\mathrm{adv}_{C_\ell}(A_2) \geq \zeta$. Fix ℓ and x, $|x| = \ell$. Define circuit C' as follows:

[1] This linear combination is the degree zero coefficient of the interpolated polynomial P'. Notice that circuit C' does not need to interpolate P' (which actually may not be possible to do by subexponential sized constant depth circuit) since the points at which values of f are given are fixed (once the random choices are fixed) and therefore, the inverse of the corresponding van der Monde matrix can simply be hardwired into C'.

Let $t = c \cdot \log \ell$ for a suitable $c > 1$. Randomly choose t strings $r_1, \ldots, r_t$ with $|r_i| = \ell$. For each non-empty subset J of $\{1, \ldots, t\}$, let $r_J = \oplus_{i \in J} r_i$ (these r_J's are pairwise independent and this is exploited in the proof). Fix s, $|s| = t$ and compute $\sigma_J = \oplus_{i \in J} s_i$ where s_i is the i^{th} bit of s. Now compute i^{th} bit of $f(x)$ as the majority of the $2^t - 1$ values (obtained by varying J) $\sigma_J \oplus C_{2\ell}(x, r_J \oplus e^i)$ where e^i is an ℓ-bit vector with only the i^{th} bit one. Finally, output the guess for $f(x)$ thus computed for *each* of the $2^t - 1$ values of s.

It was shown in [GL89] that for at least ζ^2 fraction of inputs x, $f(x)$ is present in the list of strings output by the circuit C' with probability close to one. Now there are two ways to design a circuit C'' that outputs $f(x)$ depending on the value of ζ. If $\zeta = \frac{1}{\ell^{\Omega(1)}}$ then C'' randomly picks one string output by C' and outputs it. The probability that it succeeds is close to $\frac{1}{2^t - 1} = \frac{1}{\ell^{O(1)}}$. Fix the internal random bits used by this circuit by averaging. The resulting circuit correctly outputs $f(x)$ on at least ζ^3 fraction of inputs.

The second way is for $\zeta \geq \frac{5}{6}$. In this case, C'' selects the right string from the output list of C' as follows (suggested in [Imp95]): randomly choose $O(\ell)$ many strings $r \in \{0,1\}^\ell$ and for each string u output by C' test if $u \cdot r = C_{2\ell}(x, r)$ and output the string u for which the largest number of r's satisfy the test. It was shown in [Imp95] that suitably fixing random strings r, if $f(x)$ appears in the output list then C'' would certainly output it. Therefore, the fraction of inputs on which C'' is correct is at least ζ^2.

In either case, the depth of the circuit C'' is only a constant more than of $C_{2\ell}$. Although C'' uses majority gates, they are only over $\ell^{O(1)}$ many inputs and so can be replaced by constant depth subexponential AC^0 size circuits.

Notice that the above two constructions cannot handle ζ between $\frac{1}{\ell^{o(1)}}$ and $o(1)$. However, these values of ζ are never required in the constructions[2].

2.3 Stage 3: Analyzing Impagliazzo's Hard-Core Construction

This stage has three substages. In the first substage, starting from set A_2 with $\mathrm{adv}_{2^{\epsilon_2 \ell}}(A_2) \leq 1 - \frac{1}{16\ell^3}$, set A' is constructed with $\mathrm{adv}_{2^{\epsilon' \ell}}(A') \leq 1 - \frac{1}{16\ell^2}$ for any $\epsilon' < \epsilon_2$. In the next stage, set A'' is constructed from A' with $\mathrm{adv}_{2^{\epsilon'' \ell}}(A'') \leq 1 - \frac{1}{16\ell}$ for any $\epsilon'' < \epsilon'$. And in the third substage, from A'', set A_3 is constructed with $\mathrm{adv}_{2^{\epsilon_3 \ell}}(A^3) \leq \frac{15}{16}$ for any $\epsilon_3 < \epsilon''$.

All the three substages are identical. We describe only the first one.

Construction of A'. Set A' is defined as: $rs \in A'$ iff $|r| = c \cdot \ell$, $|s| = 2\ell$ and $r \cdot g(s) = 1$ where $g(s) = A_2(x_1)A_2(x_2) \cdots A_2(x_{c \cdot \ell})$ with $x_1, \ldots, x_{c \cdot \ell}$, $|x_i| = \ell$, (for an appropriate constant c) generated from s in a pairwise-independent fashion—let $s = s_1 s_2$ with $|s_1| = |s_2| = \ell$, then $x_i = s_1 \cdot i + s_2$ in the field F_{2^ℓ}.

[2] In fact there is a third way that works for all values of ζ. However, it uses error-correcting codes and decoding these appears to require more than constant depth subexponential size circuits. So we cannot use it.

Correctness of the construction. Let a circuit family $\{C_\ell\}$ of size $2^{\epsilon'\ell}$ be given such that $\text{adv}_{C_\ell}(A') \geq 1 - \frac{1}{16\ell^2}$. First invoke the (second) Goldreich-Levin construction to conclude that there exists a circuit family $\{C'_\ell\}$ of size $2^{\delta'\ell}$ for $\epsilon' < \delta' < \epsilon_2$ that computes function $g(s)$ on at least $(1-\frac{1}{16\ell^2})^2 \geq 1 - \frac{1}{8\ell^2}$ fraction of inputs. Fix an ℓ. Define a circuit C'' as:

> On input x, $|x| = \ell$, randomly select an i, $1 \leq i \leq c \cdot \ell$. Then randomly select first half s_1 of the seed s and let $s_2 = x + s_1 \cdot i$ (this ensures that x occurs as x_i). Use s to generate $x_1, \ldots, x_{c\cdot\ell}$. Output the i^{th} bit of $C'_{c\cdot\ell^2}$ as guess for $A_2(x)$.

It was shown in [Imp95] that, for any given set $S \subset \{0,1\}^\ell$ with $|S| \geq \frac{2^\ell}{16\ell^3}$, when input x is randomly selected from S, the probability that $C''(y) = A_2(y)$ is at least $\frac{3}{5}$.

From the circuit C'', construct another circuit C''' as: take ℓ^2 copies of C'' (using different random bits for each one), and take the majority of their output values. For any x, if the probability of C''' incorrectly computing $A_2(x)$ is more than $\frac{1}{2^\ell}$ then it must be that C'' incorrectly computes $A_2(x)$ with probability more than $\frac{2}{5}$. By the above property of C'', there cannot be more than $\frac{2^\ell}{16\ell^3}$ such x's. Therefore, on at least $1 - \frac{1}{16\ell^3}$ fraction of inputs, C''' computes A_2 correctly with probability at least $1 - \frac{1}{2^\ell}$. Now fix the random bits of C''' such that the resulting circuit computes A_2 correctly on at least $1 - \frac{1}{16\ell^3}$ fraction of inputs.

As for the size and depth increase, circuit C''' (as well as the final circuit) uses one majority gate (on ℓ^2 inputs) at the top and one bottom layer of parity gates (on ℓ inputs). It also uses ℓ^2 copies of C'' in parallel. Therefore, the size of the circuit is at most $2^{\epsilon_2\ell}$ since $\epsilon_2 > \delta'$ and depth is only a constant more. This contradicts the assumption about A_2.

The above construction of circuit C''' is used again later with different parameters: starting with a circuit C'' that computes the given set with probability at least $\frac{1}{2} + \epsilon$ on any subset of strings of size $O(2^\ell)$, we can use the above construction to obtain a circuit C''' that computes the set on a constant fraction of inputs in a similar fashion. This circuit is constructed by taking the majority of $O(\frac{\ell}{\epsilon^2})$ copies of another circuit. The value of ϵ would be crucial in our calculations there.

2.4 Stage 4: Analyzing Impagliazzo-Wigderson's Construction

Construction of A_4. Set A_4 is defined as: $rs \in A_4$ iff $|r| = \ell$, $|s| = k\ell$, and $r \cdot g'(s) = 1$ where $g'(s) = A_3(x_1)A_3(x_2)\cdots A_3(x_\ell)$ with x_is generated from s via a generator whose output is XOR of the outputs of an expander graph based generator and a NW-design based generator.

Correctness of the construction. The construction is this stage is very similar to the one in previous stage. Let a circuit family $\{C_\ell\}$ of size $2^{\epsilon_4\ell}$ be given such that $\text{adv}_{C_\ell}(A_4) \geq \frac{1}{2^{\epsilon_4\ell}}$. Invoke the (first) Goldreich-Levin construction to

obtain a circuit family $\{C'_\ell\}$ of size $2^{\epsilon'\ell}$ computing function g' on $\frac{1}{2^{\epsilon'\ell}}$ fraction of inputs for $\epsilon' > \epsilon_4$.

Fix and ℓ. Construct a circuit C'' in a similar fashion (although the analysis becomes different) that computes A_3 with probability at least $\frac{1}{2} + \frac{1}{2^{\epsilon''\ell}}$ (for any $\epsilon'' > \epsilon'$) on any given set of size $\frac{2^\ell}{16}$ and then construct C''' from C'' by taking the majority of $O(2^{2\epsilon''\ell})$ copies of C''. As before, it can be shown that C''' computes A_3 correctly with probability more than $1 - \frac{1}{2^\ell}$ on all but $\frac{1}{16}$ fraction of inputs. Fixing random bits of C''' suitably gives a circuit that correctly computes A_3 on at least $\frac{15}{16}$ fraction of inputs.

The size of circuit C''' is at most $2^{\epsilon_3\ell}$ since $\epsilon_3 > \epsilon''$. The depth of C''' is still only a constant more than that of C_ℓ since the output of the generator used in construction of A_4 can be easily computed: the output of the generator gets fixed upon fixing the random bit values apart from ℓ fixed positions where the string x is written.

However, the majority gate at the top of C''' has $2^{\Omega(\ell)}$ inputs. This *cannot* be done using AC^0 circuits in constant depth and $2^{\delta n}$ size for any $\delta > 0$. In fact, this is the only place where the depth condition is violated.

2.5 Stage 5: Analyzing Nisan-Wigderson's Construction

Construction of generator. Pseudo-random generator G_n is defined as: given $k \cdot \log n$ length seed s, compute n "nearly disjoint" subsets of bit positions in the seed of size $t \cdot \log n$ each ($t < k$). Let the strings written in these positions be $x_1, \ldots, x_n$, $|x_i| = t \cdot \log n$. Output $A_4(x_1)A_4(x_2)\cdots A_4(x_n)$.

Correctness of the construction. Let C be a circuit of size n such that

$$| \operatorname{prob}_{x\in\{0,1\}^n}\{C(x) = 1\} - \operatorname{prob}_{s\in\{0,1\}^{k\cdot n}}\{C(G_n(s)) = 1\} | \geq \frac{1}{n}.$$

Define circuit C^i as:

> On input x_i *and* $A_4(x_1)\cdots A_4(x_{i-1})$, randomly select a bit b and a string r of length $n - i$. Compute $o = C(A_4(x_1)\cdots A_4(x_{i-1})br)$. Output $b \oplus o$.

It was shown in [NW94] that for at least one i, C^i correctly computes $A_4(x_i)$ on at least $\frac{1}{2} + \frac{1}{n^2}$ fraction of inputs.

Exploiting the property that the subsets of bit positions determining each of $x_1, \ldots, x_{i-1}$ are nearly disjoint from those determining x_i, one can fix the random bits of C^i and of the seed s except for those bits that determine x_i such that the advantage of C^i in computing $A_4(x_i)$ is preserved *and* the value of $A_4(x_j)$ (for $j < i$) is needed by the circuit (as x_i varies) for at most n different inputs. So all values of A_4 needed by the circuit (at most n^2) can be hardwired into it, thus eliminating the need of providing $A_4(x_1)\cdots A_4(x_{i-1})$ as part of the input.

Let the final circuit be C''. The size of C'' is $O(n^2)$ and $\mathrm{adv}_{C''}(A_4) > \frac{1}{n^2}$ on inputs of size $t \cdot \log n$. For a suitable choice of t and k, this contradicts the

hardness of A_4. The depth of C'' is only a constant more than the depth of C as the only additional computation needed is to select the correct hardwired values of $A_4(x_1), \ldots, A_4(x_{i-1})$ depending on the input x_i (this is a simple table lookup).

2.6 Analyzing Constructions of Sudan-Trevisan-Vadhan

The above bottleneck prompts us to look at other constructions of true pseudo-random generators present in the literature: there are two such constructions known given in [STV99]. However, both these constructions have similar bottlenecks. We point out these bottlenecks below:

First construction. This construction uses a *false entropy generator*. This generator makes use of the hard-core result of Impagliazzo [Imp95]. The value of ϵ that the construction requires in the hard-core result is $\frac{1}{2^{O(\ell)}}$. So this has the same problem as the construction of [IW97]: it requires to compute the majority of $2^{O(\ell)}$ bits.

Second construction. This construction actually shows that stage 3 and 4 above can be *bypassed*. In other words, the multivariate polynomial P has enough redundancy to directly ensure that no circuit family of size $2^{\beta\ell}$ can compute the function on more than $\frac{1}{2^{\beta\ell}}$ fraction of inputs. However, the proof for this result is far more involved than the proof of stage 1. In the proof, to interpolate the polynomial correctly on a random line, at least $2^{\kappa\ell}$ samples are needed. This requires, amongst other things, xoring of $2^{\kappa\ell}$ bits and also computing $2^{\kappa\ell}$th power of a given element in a field of size $2^{O(\ell)}$. None of these can be performed by constant depth $2^{O(\ell)}$ sized AND-OR circuits.

3 Proof of Theorem 1

The problem in working with AC^0 circuits is that they are too weak to do even simple computations. But we can use this drawback to our advantage! Since good lower bounds for AC^0 circuits are known [Has86], one can construct unconditional pseudo-random generator against such circuits. In [Nis91], Nisan used lower bounds on parity function to obtain pseudo-random generators against depth d, size n AC^0 circuits that stretch seeds of size $(\log n)^{O(d)}$ to n bits. Moreover, each output bit of these generators is simply parity of some of the seed bits. Therefore, each output bit of the generator can be computed by an AC^0 circuit of size $n^{o(1)}$ and depth $O(d)$.

So, given an AC^0 circuit C of depth d and size n that accepts δ fraction of inputs, when we combine this circuit with the pseudo-random generator of [Nis91], we get *another* AC^0 circuit of depth $O(d)$ and size $O(n^2)$ that has only $(\log n)^{O(d)}$ input bits and still accepts $\delta \pm \frac{1}{n}$ fraction of inputs. Let us try to construct a true pseudo-random generator against *such* a circuit using the Nisan-Wigderson construction. This generator needs to stretch $O(\log n)$ bits to $(\log n)^{O(d)}$ bits. If we examine the Nisan-Wigderson construction of the generator, it is apparent

that—if we fix the approximation error to $\frac{1}{(\log n)^{O(1)}}$ instead of $\frac{1}{n}$—such a generator can be constructed *provided* there exists a set $A \in \mathrm{E}$ such that for any depth $O(d)$ circuit family $\{C_\ell\}$ of size $2^{\delta\ell}$, $\mathrm{adv}_{C_\ell}(A) < \frac{1}{\ell^{O(d)}}$. Now notice that such a set can be easily constructed by modifying the stage 4 of the construction! Since instead of $\epsilon = \frac{1}{2^{O(\ell)}}$ we now have $\epsilon = \frac{1}{\ell^{O(d)}}$, the majority gate needed in the construction will have a fan-in of only $\ell^{O(d)}$, and this *can* be done by constant depth AND-OR circuits of subexponential size[3]. Hence the overall construction now becomes six stage one: first three stages are identical to the ones described above; the fourth stage is modified for weaker approximation needed; the fifth stage uses Nisan-Wigderson construction for pseudo-random generator that stretches the seed only polynomially; this stretched seed acts as seed for the Nisan generator in the final stage that stretches the output to n bits.

It is interesting to note that each output bit of this pseudo-random generator is simply an XOR of several bits of the characteristic function A_1: the multivariate polynomial construction in Stage 1 is just an XOR of some input bits; Stage 2, 3, and 4 constructions are clearly simple XORs (computing which bits to XOR requires some effort though); the fifth stage merely copies some bits from input to output; and the last stage (it uses parity function) is also xoring some bits.

Acknowledgements

An anonymous referee's suggestions have helped to make the writeup more readable. Eric Allender and Adam Klivans pointed out a wrong claim made in an earlier version of the paper.

References

BFNW93. L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

BM84. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

GL89. O. Goldreich and L. A. Levin. A hardcore predicate for all one-way functions. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

Has86. J. Hastad. *Computational limitations on small depth circuits.* PhD thesis, Massachusetts Institute of Technology, 1986.

Imp95. R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

IW97. R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[3] Alternatively, one can use a construction (given in [Imp95]) that decreases the advantage to $\frac{1}{\ell^{O(1)}}$ using k-wise independent generation of strings. This avoids the construction of [IW97] altogether.

KL83. R. M. Karp and M. Luby. Monte-Carlo algorithms foe enumeration and reliability problems. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 56–64, 1983.

Nis91. N. Nisan. Pseudo random bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

NW94. N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. Sys. Sci.*, 49(2):149–167, 1994.

STV99. M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 537–546, 1999.

Yao82. A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

The First-Order Isomorphism Theorem

Manindra Agrawal

Department of Computer Science
IIT Kanpur
Kanpur 208016, India
manindra@iitk.ac.in

Abstract. For any class $\mathcal{C}$ und closed under NC^1 reductions, it is shown that all sets complete for $\mathcal{C}$ under first-order (equivalently, Dlogtime-uniform AC^0) reductions are isomorphic under first-order computable isomorphisms.

1 Introduction

One of the long-standing conjecture about the structure of complete sets is the isomorphism conjecture (proposed in [BH77]) stating that all sets complete for NP under polynomial-time reductions are polynomial time isomorphic. As the conjecture cannot be resolved either way unless we discover non-relativizable techniques (see [KMR88,KMR89,FFK92] for more details), efforts have been made to prove the conjecture in restricted settings by restricting the power of reductions (see for example [Agr96,AAR98]). One of the most natural definition of restricted reductions is that of functions computed by *uniform constant-depth* (or AC^0) circuits (first studied in [CSV84]). These reductions provide the right notion of completeness for small complexity classes (logspace and below). Also, it has been observed that *natural* complete problems for various complexity classes remain complete under such reductions [IL95,Imm87]. Although the class of AC^0 functions is much smaller than the class of polynomial-time functions, it is interesting to note that, till recently, there was *no* known example of an NP-complete set that is not complete under uniform AC^0 reductions [AAI+97].

The notion of uniformity to be used with AC^0 circuits is widely accepted to be that of *Dlogtime-uniformity* (see Section 3 for definition). Under this uniformity condition, these circuits admit a number of different characterizations [BIS90,AG91]: functions computed by first-order logic formulae [Lin92], $O(1)$-alternating log-time TMs [Sip83], logspace rudimentary predicates [Jon75] etc.

The isomorphism conjecture for complete sets for NP under AC^0 reductions has been studied before. Allender et. al. [ABI93] showed that all sets complete under *first-order projections* (these are very simple functions computed by uniform circuits with no gates [IL95]) are Dlogtime-uniform AC^0-isomorphic (i.e., the isomorphism between any two such sets is computable in both directions by Dlogtime-uniform AC^0 circuits). This was improved in [AAR98] who showed that all sets complete under u-uniform (for any u) AC^0 reductions are non-uniform

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 70–82, 2001.
© Springer-Verlag Berlin Heidelberg 2001

AC^0-isomorphic. Notice that this result proves the isomorphism conjecture for non-uniform AC^0 reductions but not for Dlogtime-uniform reductions. The uniformity condition for isomorphisms was improved first in [AAI+97] to P-uniform and then in [Agr01] to logspace-uniform thus proving the isomorphism conjecture for P-uniform and logspace-uniform AC^0 reductions respectively. More specifically, for all sets complete under u-uniform AC^0 reductions, [AAI+97] shows that they are (u+P)-uniform AC^0-isomorphic, while [Agr01] shows that they are (u+logspace)-uniform AC^0-isomorphic. However, the conjecture remains open for Dlogtime-uniform AC^0 reductions, which is, in many ways, the correct formulation of the isomorphism conjecture for constant depth reductions.

In this paper, we prove that all complete sets for NP under u-uniform AC^0 reductions are (u+Dlogtime)-uniform AC^0-isomorphic thus proving the isomorphism conjecture for constant depth reductions. Since there are a number of alternative characterizations of Dlogtime-uniform AC^0 circuits, this theorem can be viewed in many interesting ways, e.g., *all sets complete under first-order reductions are first-order isomorphic* (first-order functions are computed by first-order formulae). The above in fact holds for any class closed under TC^0 reductions.

The next section provides an outline of our proof. Section 3 contains definitions, and the subsequent sections are devoted to proving the result.

2 Proof Outline

The overall structure of the proof remains as given in [AAR98]. The proof in [AAR98] is a three stage one:

Stage 1 (Gap Theorem): This shows that all complete sets under u-uniform AC^0 reductions are also complete under *non-uniform* NC^0 reductions. This step is non-uniform.

Stage 2 (Superprojection Theorem): This proves that all complete sets under u-uniform NC^0 reductions are also complete (u+P)-uniform *superprojections*, where superprojections are functions similar to projections. This step is P-uniform.

Stage 3 (Isomorphism Construction): This proves that all complete sets under u-uniform superprojections are isomorphic under (u+Dlogtime)-uniform AC^0 isomorphisms. This step is Dlogtime-uniform: starting with Dlogtime-uniform superprojections, one gets Dlogtime-uniform AC^0 isomorphisms.

The proof of Gap Theorem uses the Switching Lemma of [FSS84] in the construction of NC^0 reductions and is the reason for its non-uniformity. In [AAI+97] the lemma was derandomized using method of conditional probabilities making the stage P-uniform. Improving upon this, in [Agr01], the lemma was derandomized by constructing an appropriate *pseudo-random generator*. This made the stage logspace-uniform.

The Superprojection Theorem of [AAR98] uses the Sunflower Lemma of [ER60] which is P-uniform. This construction was replaced in [Agr01] by

a probabilistic construction that could be derandomized via an appropriate pseudo-random generator. This again resulted in a logspace-uniform construction.

Clearly, the uniformity of both these stages needs to be improved to obtain Dlogtime-uniformity. It is useful to note here that we need to make both the stages AC^0-uniform only as that makes the isomorphism constructed by Stage 3 also AC^0-uniform and then the AC^0 circuit used in uniformity can be incorporated in the AC^0 circuit for the isomorphism making the resulting AC^0 circuit Dlogtime-uniform. In fact this is the best that we can hope to do as it is known that the Gap Theorem *cannot* be made Dlogtime-uniform [AAR98].

We preserve the idea of [Agr01] of first giving a probabilistic construction and then derandomizing it via an appropriate pseudo-random generator in both the stages. The improvement in the uniformity condition of first stage is achieved by a careful construction of the pseudo-random generator needed that allows it to become AC^0-uniform. The second stage presents a bigger problem. We replace the probabilistic construction used in [Agr01] by a more involved probabilistic construction and then derandomize it to obtain AC^0-uniformity.

Combining the above constructions together with the Isomorphism Construction, we get Dlogtime-uniform AC^0-isomorphisms.

3 Basic Definitions and Preliminaries

We assume familiarity with the basic notions of many-one reducibility as presented, for example, in [BDG88].

A *circuit family* is a set $\{C_n : n \in \mathbf{N}\}$ where each C_n is an acyclic circuit with n Boolean inputs $x_1, \ldots, x_n$ (as well as the constants 0 and 1 allowed as inputs) and some number of output gates $y_1, \ldots, y_r$. $\{C_n\}$ has *size* $s(n)$ if each circuit C_n has at most $s(n)$ gates; it has *depth* $d(n)$ if the length of the longest path from input to output in C_n is at most $d(n)$.

For a circuit family $\{C_n\}$, the *connection set* of the family is defined as:

$$\mathrm{Conn}_C = \{\langle n, t, i, j\rangle \mid \text{ gate } i \text{ in } C_n \text{ is of type } t \text{ and takes input from gate } j\}.$$

A family $\{C_n\}$ is *u-uniform* if the connection set can be computed by a machine (or circuit) with a resource bound of u. In this paper, we will consider two notions of uniformity: Dlogtime-uniformity [BIS90] and AC^0-uniformity. In the first, the connection set is computed by a TM with random access tapes working in $O(\log n)$ time (which is linear time as a function of input size), and in the second, the connection set is computed by an AC^0 circuit of polynomial size (which is exponential size in terms of input size). We will follow the standard convention that whenever the connection set is computed by a circuit family, the circuit family is assumed to be Dlogtime-uniform. So, for example, AC^0-uniform means that the set can be computed by a Dlogtime-uniform AC^0 family of circuits.

A function f is said to be in AC^0 if there is a circuit family $\{C_n\}$ of size $n^{O(1)}$ and depth $O(1)$ consisting of unbounded fan-in AND and OR and NOT gates

such that for each input x of length n, the output of C_n on input x is $f(x)$. We will adopt the following specific convention for interpreting the output of such a circuit: each C_n will have $n^k + k\log(n)$ output bits (for some k). The last $k \log n$ output bits will be viewed as a binary number r, and the output produced by the circuit will be binary string contained in the first r output bits. It is easy to verify that this convention is AC^0-equivalent to any other reasonable convention that allows for variable sized output, and for us it has the advantage that only $O(\log n)$ output bits are used to encode the length.

With this definition, the class of Dlogtime-uniform AC^0-computable functions admits many alternative characterizations, including expressibility in first-order with $\{+, \times, \leq\}$, [Lin92,BIS90] the logspace-rudimentary reductions of Jones [Jon75,AG91], logarithmic-time alternating Turing machines with $O(1)$ alternations [BIS90] and others. This lends additional weight to our choice of this definition.

NC^0 is the class of functions computed in this way by circuit families of size $n^{O(1)}$ and depth $O(1)$, consisting of fan-in two AND and OR and NOT gates. Note that for any NC^0 circuit family, there is some constant c such that each output bit depends on at most c different input bits. An NC^0 function is a *projection* if its circuit family contains no AND or OR gates. For the sake of simplicity, we assume that NC^0 and projection functions *do not have variable sized output*. This may seem restrictive at a first glance, however, as we show later, that at least for complete sets we can ensure this property.

For a complexity class $\mathcal{C}$, a $\mathcal{C}$-isomorphism is a bijection f such that both f and f^{-1} are in $\mathcal{C}$. Since only many-one reductions are considered in this paper, a "$\mathcal{C}$-reduction" is simply a function in $\mathcal{C}$.

(A *language* is in a complexity class $\mathcal{C}$ if its characteristic function is in $\mathcal{C}$. This convention allows us to avoid introducing additional notation such as FAC^0, FNC^1, etc. to distinguish between classes of languages and classes of functions.)

4 AC^0-Uniform Gap Theorem

In this section, we prove the AC^0-uniform version of the Gap Theorem of [AAR98]:

Theorem 1. *For any class $\mathcal{C}$ closed under NC^1 reductions, all complete sets for $\mathcal{C}$ under u-uniform AC^0 reductions are also complete under $(u + \mathrm{AC}^0)$-uniform NC^0 reductions.*

Proof. We begin by outlining the proof in [AAR98] and improvements of [Agr01] as we make use of both of them.

Fix a set A in $\mathcal{C}$ that is complete under u-uniform AC^0 reductions and let $B \in \mathcal{C}$ be an arbitrary set. We need to show that B reduces to A via a $(u+\mathrm{AC}^0)$-uniform NC^0 reduction. We first define a set $\hat{B}$, which is a highly redundant version of B, as accepted by the following procedure:

> On input y, let $y = 1^k 0 z$. Reject if k does not divide $|z|$. Otherwise, break z into blocks of k consecutive bits each. Let these be $u_1 u_2 u_3 \cdots u_q$.

For each i, $1 \leq i \leq q$, let v_i be the parity of bits in u_i. Accept iff $v_1 v_2 \cdots v_q \in B$.

As one can readily observe, corresponding to a string in B there are infinitely many strings in $\hat{B}$. Also, $\hat{B}$ reduces to B via an NC^1 reduction and so $\hat{B} \in \mathcal{C}$. Fix a reduction of $\hat{B}$ to A given by u-uniform AC^0 circuit family $\{C_n\}$, say. Now define a reduction of B to $\hat{B}$ as follows (it would be useful to keep the above definition of $\hat{B}$ in mind while reading this definition):

Given an input x, $|x| = n$, let $m = n^t$ for an appropriate constant t to be fixed later. Consider the circuit $C_{m/n+1+m}$ with the first $m/n+1$ bits set to $1^{m/n}0$ resulting in circuit C'_m, say. Apply the Switching Lemma of [FSS84] on C'_m to obtain a setting of all but $\Omega(n \cdot (\log n)^2)$ input bits such that the circuit reduces to an NC^0 circuit and in addition, all the n blocks of $m/n = n^{t-1}$ consecutive bits in the input have at least $(\log n)^2$ unset bits (it was shown in [AAR98] that this can be ensured and this is what governs the choice of constant t). Now set all those unset bits to zero that influence at least one of the last $k \cdot \log n$ bits of the output (remember that these bits encode the length of the output as per our convention). This sets $O(\log n)$ additional unset bits. Since each block had $(\log n)^2$ unset bits to begin with, each block would still have at least two unset bits. Now for each of the n blocks, set all but one bits of the block to ensure that the number of ones in the block is 0 modulo 2 (this can also always be done as there is at least one unset bits available for setting). This sets all the m bits of input to C'_m except for n bits and on these n unset bits the circuit C'_m becomes an NC^0 circuit. Now map x to a string of length $m/n + 1 + m$ whose first $m/n + 1$ bits are set to $1^{m/n}0$ and the remaining bits are set according to the above procedure and the i^{th} remaining unset bit is given the value of i^{th} bit of x.

It is easy to verify that the mapping constructed above is indeed a reduction of B to $\hat{B}$. Notice that this reduction is simply a *projection*: each input bit is mapped to some output bit directly and there are no gates in the circuit computing the reduction. It is also clear that a composition of this reduction with the reduction of $\hat{B}$ to A is a reduction of B to A that can be computed by an NC^0 circuit family. The uniformity machine (or circuit) for this NC^0 circuit family is required to do the following tasks, apart from generating the circuit C'_m itself:

1. identity the settings of input bits to circuit C'_m that make the circuit an NC^0 circuit,
2. given such a setting, transform the circuit C'_m to the equivalent NC^0 circuit, and
3. set some of the unset bits as outlined above to leave only one unset bit in each block (in which string x would be placed).

The second task can be done by a Dlogtime-uniform AC^0 circuit that, for each output bit of the circuit C'_m, guesses the $O(1)$ input bits influencing the corresponding NC^0 circuit and then verifies this guess by evaluating C'_m on all

possible settings of these bits and noting if the chosen output bit of C'_m becomes constant for each setting or not.

For the third task, a Dlogtime-uniform AC^0 circuit can identify which unset bits influence the output bits coding length of the output, however, to set bits in a block appropriately (so that number of ones is 0 modulo 2), one requires a parity gate making the overall circuit an NC^1 circuit.

For the first task, we first note that according to the Switching Lemma of [FSS84] *most of the settings work*. In [AAI+97], a polynomial-time algorithm was given to identify one such setting given the circuit C'_m thus making the NC^0 circuit P-uniform. In [Agr01], a pseudo-random generator was constructed that stretches a seed of $O(\log n)$ bits to m bits such that on most of the strings output by the generator the circuit C'_m reduces to an NC^0 circuit. Using this, a uniformity machine can be constructed that first generates all possible $n^{O(1)}$ outputs of the generator and then, in parallel, checks which one of these is "good" by attempting to transform C'_m to an NC^0 circuit as outlined above. The power of the machine is decided by the difficulty of computing the generator. In [Agr01], the generator designed can be computed in logspace making the entire construction logspace-uniform.

To obtain AC^0-uniformity, we need to improve upon both the first and third tasks. For the first task, one can try to obtain a generator that is computable by a Dlogtime-uniform AC^0 circuit. However, improving the third one seems impossible at the first glance as it is well known that computing parity of n bits cannot be done by even non-uniform AC^0 circuits [FSS84]. We solve this problem be a clever design of the generator: the generator would be such that it associates a sign (0 or 1) with each unset bit and the parity of all the set bits and signs of unset bits in a block is always zero! This trivializes the third task. The reduction of B to $\hat{B}$ has to be changed slightly to make this work: map the i^{th} bit of x to the i^{th} unset bit if its sign is 0, else map it to the i^{th} unset bit by first complementing it.

We now give the generator construction of [Agr01] and then show how to improve it so that both the first and third tasks are completed. The generator is a combination of two types of primitive generators: (1) generators that produce bits that are $\frac{1}{n^{O(1)}}$-biased, $O(\log n)$-wise independent [NN90], and (2) generators, based on Nisan-Wigderson designs [NW94]. The generator consists of ℓ primitive generators of each type where ℓ is a constant dependent on the depth and size of the circuit C'_m. Also, each one of these primitive generators requires seed of length $O(\log n)$, and therefore, the seed length of the generator is $O(\log n)$. The generator is constructed as follows:

> Let $G^1_{IND}, \ldots, G^\ell_{IND}$ be ℓ primitive generators producing bits that are $\frac{1}{n^{O(1)}}$-biased, $O(\log n)$-wise independent, and $G^1_{NW}, \ldots, G^\ell_{NW}$ be primitive generators based on NW-designs (their construction will be discussed later). The i^{th} bit of the generator G of [Agr01] is computed as: write i in binary and let $i = i_1 i_2 \cdots i_{\ell+1}$ where $|i_j| = \frac{|i|}{2^j}$ for $1 \leq j \leq \ell$ (we assume $|i|$ to be a power of two to avoid complications). Compute bit $G^j_{NW}[i_j i_{j+1} \cdots i_{\ell+1}]$ (we use $G[k]$ to denote the k^{th} bit of function G)

for $1 \leq j \leq \ell$. Let j_0 be the first j for which $G^j_{NW}[i_j i_{j+1} \cdots i_{\ell+1}]$ is zero. If there exists such a j_0 then let $G[i] = \oplus_{1 \leq j \leq j_0} G^j_{IND}[i_j i_{j+1} \cdots i_{\ell+1}]$. If there is no such j_0 then leave $G[i]$ unset and compute its sign as $\oplus_{1 \leq j \leq \ell} G^j_{IND}[i_j i_{j+1} \cdots i_{\ell+1}]$.

The following lemma was proved for this generator in [Agr01][1]:

Lemma 1. *[Agr01] Let C be any* AC^0 *circuit of a depth and size bounded by C'_m having m input bits. Then on at least half of the outputs of G, C reduces to an* NC^0 *circuit.*

It is clear from the construction of G above, that the computational resources required for G depend on the resources required to compute the two types of primitive generators. In particular, if both these types of primitive generators can be computed by Dlogtime-uniform AC^0 circuits, the generator G can also be computed by such circuits.

Let us now see constructions for these primitive generators. Three simple constructions of $\frac{1}{n^{O(1)}}$-biased, $O(\log n)$-wise independent generators are given in [AGHP90]. We choose one based on quadratic residues in a small field (in [Agr01] a different generator is used): i^{th} bit of $G^j_{IND}(s^j)$ is 1 iff the number $s^j + i$ is a quadratic non-residue in the field F_p with prime $p = n^{O(1)}$ (here s^j is the seed). This can be done by a Dlogtime-uniform AC^0 circuit: first an appropriate prime p is computed (fixing the field F_p), then s^j is added to i (addition is modulo p), and finally it is checked if there exists an x such that $x^2 = s^j + i$. All these computations can be done by Dlogtime-uniform AC^0 circuits as the field size is small (as shown in [BIS90]).

The generator G^j_{NW} is defined as: let $i = i_1 i_2$ with $|i_1| = |i_2| = k$; let seed $s^j = s^j_1 s^j_2 \cdots s^j_c$ with $|s^j_1| = |s^j_2| = \cdots = |s^j_c| = k$ for appropriate constant c; compute $i' = \sum_{e=1}^{c} s^j_e \cdot (i_2)^{e-1}$ where all the operations are over field F_{2^k}, and set the i^{th} bit to 1 iff $i' = i_1$. Again, all the computations here can be done by a Dlogtime-uniform AC^0 circuit (shown in [BIS90]).

Thus the generator G can be computed by a Dlogtime-uniform AC^0 circuit. The primitive generator G^1_{NW} sets exactly one bit to 1 in consecutive blocks of $m^{\frac{1}{2}}$ bits, the generator G^2_{NW} sets exactly one bit to 1 in consecutive blocks of $m^{\frac{1}{4}}$ unset bits remaining, etc. Thus the generator G leaves exactly unset one bit in consecutive blocks of $m^{\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^\ell}} = m^{1-\frac{1}{2^\ell}}$ bits which makes a total of $m^{\frac{1}{2^\ell}}$ unset bits. Now choosing $t = 2^{\ell+1}$ (recall that $m = n^t$) ensures that each of the n blocks contains exactly $n \geq (\log n)^2$ unset bits as required.

However, this generator *does not* guarantee that the parity of all the set bits and signs of the unset bits in a block is zero (the length of a block is $m/n = n^{2^{\ell+1}-1}$). To achieve this, we change the definition of generators G^j_{IND} in such a way that parity of all the bits that it contributes to setting of bits and signs

[1] Actually, in [Agr01] the generator construction is slightly different: $G[i]$ is simply set to $G^{j_0}_{IND}[i_{j_0} \cdots i_{\ell+1}]$ when j_0 exists. However, the lemma holds for this modification too, and this modification makes our current construction simpler.

in a block is zero. Notice that the generator G^j_{IND} contributes exactly $n^{2^{\ell+2-j}-1}$ bits to a block. Change the generator G^j_{IND} by replacing every $(f \cdot \log n)^{th}$ bit by the parity of previous $f \cdot \log n - 1$ bits for a large enough constant f (f should be chosen so that it is power of two and the generator is required to be $f' \cdot \log n$-wise independent for $f' < f$). Without loss of generality, we can assume that n is of the form 2^{2^u} and then, since f is a power of two, $f \cdot \log n$ would divide n. This ensures that the parity of all the bits contributed by the modified generator to a block is zero. However, we now need to show that the modified generator is still $\frac{1}{n^{O(1)}}$-biased, $O(\log n)$-wise independent. This follows immediately from the fact that any set of $< f \cdot \log n$ bits of the modified generator is still $\frac{1}{n^{O(1)}}$-biased (follows from [AGHP90]), and therefore these bits are independent with a similar bias (shown in [NN90]). Thus with these modified primitive generators, the generator G satisfies all the required conditions proving the theorem. □

5 AC^0-Uniform Superprojection Theorem

We start with the definition of a superprojection [AAR98].

Definition 1. *An NC^0 reduction $\{C_n\}$ is a* superprojection *if the circuit that results by deleting zero or more of the output bits in each C_n is a projection wherein each input bit (or its negation) is mapped to some output.*

Now we prove the AC^0-uniform Superprojection Theorem:

Theorem 2. *For any class $\mathcal{C}$ closed under NC^1 reductions, all complete sets for $\mathcal{C}$ under u-uniform NC^0 reductions are also complete under $(u + AC^0)$-uniform superprojections.*

Proof. Fix a set A in $\mathcal{C}$ that is complete under u-uniform NC^0 reductions and let $B \in \mathcal{C}$ be an arbitrary set. We need to show that B reduces to A via a $(u+AC^0)$-uniform superprojection. We first define, as before, a set $\hat{B}$ as accepted by the following procedure:

> On input y let $y = z'11z$ such that $z' \in \{00, 01, 10\}^*$. Break z' into pairs of bits. Ignoring all the 00 pairs, consider the first $\log |z|$ pairs. Define number k by setting i^{th} bit of k to 1 if the i^{th} of the above $\log |z|$ pairs is 10, to 0 otherwise. Reject if k does not divide $|z|$. Else, break z into blocks of k consecutive bits each. Reject if the number of blocks is not a multiple of four. Else, let $z = u_1u_2u_3 \cdots u_{4q}$ with $|u_i| = k$. Let v_i be the parity of bits in u_i. Let $w_i = v_{4i-3}v_{4i-2}v_{4i-1}v_{4i}$ for $1 \leq i \leq q$ (so each w_i is a four bit string). If $w_i = 1111$ for any $1 \leq i \leq q$, accept. Else if some w_i has exactly three ones, reject. Else, for each i, $1 \leq i \leq q$, let $b_i = 1$ if w_i has exactly two ones, $b_i = 0$ if w_i has exactly one one, $b_i = \epsilon$ otherwise. Accept iff $b_1b_2 \cdots b_q \in B$.

The definition of set $\hat{B}$ is more complicated that the previous one. Even the block size ($= k$) is coded in the string in a non-straightforward way. We refer to

the bits of z' of any instance y of $\hat{B}$ as *length encoder bits* and to the bits of z as *string encoder bits*. It is easy to see that $\hat{B}$ reduces to B via an NC^1 reduction and so $\hat{B} \in \mathcal{C}$. Fix a reduction of $\hat{B}$ to A given by u-uniform NC^0 circuit family $\{C_n\}$, say. Let each output bit of any circuit C_n depend on at most c input bits.

As before, we now define a reduction of B to $\hat{B}$. The idea is same: for an appropriate m and ℓ, consider the circuit $C_{\ell+2+m}$. Set some of the input bits of $C_{\ell+2+m}$ so that the circuit on remaining unset bits is a superprojection. Now set some more bits (including all of length encoder bits) to satisfy all the conditions in the definition of set $\hat{B}$ and finally map string x to remaining unset bit positions. In [Agr01], a simple random construction for this was given and then the construction was derandomized using an appropriate generator. However, the random construction did not guarantee that in every block at least one unset bit would be present after all the settings (that is why we need to have "empty" blocks for which $b_i = \epsilon$). This makes the mapping of bits of x difficult as we need to use threshold gates to find the i^{th} unset bit. This was not the case in the previous proof as every block there had an unset bit and so the i^{th} unset bit can be identified by using an AC^0 circuit on the bits of the i^{th} block.

We give a different construction to solve this problem. Interestingly, our construction uses the central idea of the Switching Lemma proof of [FSS84] which is also used in the construction in Gap Theorem.

We first discuss a simple idea (one that is used in [Agr01]) and see why it does not work.

> Consider circuit $C_{\ell+2+m}$. Randomly set every input bit of the circuit to 0 or 1 with probability $\frac{1}{4}$ each leaving it unset with probability $\frac{1}{2}$. Say that an input bit in string encoder part is *good* if it remains unset and there is at least one output bit that now depends *only on this bit*. For any input bit that influences some output bit in $C_{\ell+2+m}$, the probability that this bit is good is at least $\frac{1}{2} \cdot (\frac{1}{4})^{c-1} > \frac{1}{4^c}$. Therefore, the expected number of good input bits is $\Omega(m')$ where m' is the number of input bits in the string encoder part of $C_{\ell+2+m}$ that influence at least one output bit. Identify all the good bits and set all the other unset input bits appropriately. This makes the circuit $C_{\ell+2+m}$ on the remaining unset bits a superprojection.

The above construction yields $\Omega(m)$ good bits provided we can ensure that nearly all the input bits influence the output (part of the complexity in definition of $\hat{B}$ is due to this requirement). The construction can easily be derandomized by using a $2c$-wise independent generator for selecting unset bits and setting remaining bits. However, the problem pointed out earlier—it cannot be ensured that every block has at least one unset bit—remains.

In our construction, we *do* use the above construction, but only after making sure that every block is guaranteed to have at least one unset bit. For this, we successively shrink the block size simultaneously making "bad" blocks (i.e., those that do not have unset bits) "empty." This is why we cannot fix the block size in the beginning of the construction unlike the previous proof.

Let x, $|x| = n$, be an instance of B. Let $m = (4n^2)^c$. Consider the circuit $C_{4^{4c}\log m+2+m}$. To begin with, set the bit numbers $4^{4c}\log m+1$ and $4^{4c}\log m+2$ of the input to $C_{4^{4c}\log m+2+m}$ to 1 identifying the first $4^{4c}\log m$ bits as length encoder and the last m bits as string encoder bits. Let C be the resulting circuit.

Split the string encoder bits of the input to C into $4n$ block of equal size $(= n \cdot (4n^2)^{c-1})$. Firstly, we notice that *every bit in every block* must influence some output bit. Suppose not. Let such a bit belong to $(4i + j)^{th}$ block. Set all the bits in all the blocks except for block numbers $4i + 1$ through $4i + 4$ so that the parity of bits in every block is zero. Set bits in blocks $4i + 1$ through $4i + 4$ except those in block $4i + j$ such that parity of bits in these blocks is one. Set all the bits in the block $4i + j$ except the the bit that does not influence any output bit so that the parity of set bits is zero. This fixes the output of circuit C. However, the value of the lone unset bit decides whether the input string belongs to the set $\hat{B}$ or not, contradicting the fact that family $\{C_n\}$ computes a reduction of $\hat{B}$ to A.

Apply a random restriction to input bits of C as outlined above using a $\frac{1}{n^2}$-biased, $4^{3c}\log n$-wise independent source to generate the restriction (instead of a $2c$-wise independent generator—the reason for this would be clear soon). There are two cases that can arise now:

Case 1. For every seed value of the generator, there is at least one block with no good bit.

Case 2. There is a seed of the generator that leaves at least one good bit in every block.

We tackle Case 1 first. Undo the above random restriction. Divide each block into n sub-blocks of equal size $(= (4n^2)^{c-1})$. For each sub-block, do the following experiment: set all other bits in all the other sub-blocks and blocks to zero, and then see if by setting an additional $4^{3c}\log n$ length encoder bits to zero all the output bits of the circuit C now depend only on at most $c - 1$ unset bits. We show later that there *must* exist such a sub-block. Fix any such sub-block. Set all bits in all other sub-blocks to zero and also those length encoder bits identified for this sub-block resulting in a circuit whose every output bit depends only on at most $c - 1$ input bits. For each length encoder bit set to zero, set its paired bit also to zero (rendering these pairs ineffectual for encoding length). We now are left with exactly $(4n^2)^{c-1}$ unset string encoder bits and at least $4^{4c}\log n - 2 \cdot 4^{3c}\log n$ unset length encoder bits.

Apply the random restriction on these bits and repeat the same process. If Case 1 keeps occurring, after $c - 1$ iterations, we would be left with exactly $4n^2$ unset string encoder bits and at least $(4^{4c}-2(c-1)\cdot 4^{3c})\cdot\log n \geq 2c\cdot\log 2n = \log m$ length encoder bits. And the circuit C is simply a projection on these unset bits! We can now fix the length encoder bits to code the block length as 1, set all the remaining length encoder bits to zero, set all but first $4n$ of unset string encoder bits to zero, set last three bits in every group of 4 unset bits to 100 and map the i^{th} bit of x to the first bit of i^{th} group. This defines a projection reduction of B to $\hat{B}$ and on outputs of this reduction, circuit C is also a projection. Therefore,

their composition is a projection (we shall see later that this composition can be computed by an AC^0-uniform circuit).

The other possibility is that after some iterations, Case 2 occurs. In that case, identify the seed on which the generator output leaves at least one good bit in each block. Set the length encoder bits to code the current block length (we argue later that this can always be done). In every block, set all the bits except one good bit to zero. Now map the string x to these good bits as above. Use the modified generator G^1_{IND} of previous proof so that the parity of all the set bits plus the signs in each block is zero (to incorporate the sign, we just need to xor it with the settings of all the unset bits). This defines a projection reduction of B to $\hat{B}$ whose composition with C is a superprojection.

There are two things remaining to be done: (1) we need to show that when Case 1 occurs then there exists a sub-block with the desired properties and when Case 2 occurs then there are enough unset length encoder bit pairs are available, (2) we need to show that the above construction can be done by a Dlogtime-uniform AC^0 circuit. The second is easy to show: we have already seen that the generator output can be computed by Dlogtime-uniform AC^0 circuit. The remaining tasks can easily be done be a Dlogtime-uniform AC^0 circuit.

To show the first, we make use of the central idea in [FSS84]. For j^{th} block, let $o_1, \ldots, o_p$ be *all* the output bits of C that depend on some bit in the block. For output bit o_i, let I_i be the set of input bits that influence o_i. Clearly, $|I_i| \leq c$. On a random restriction as defined above, the probability that a bit in j^{th} block belonging to I_i becomes good due to I_i is at least $\frac{1}{4^c}$. Let MaxSet be any *maximal* set of disjoint I_is. If $|\text{MaxSet}| \geq 4^{2c} \log n$ then drop some I_is from it to make $|\text{MaxSet}| = 4^{2c} \log n$. Since the restriction bits are $4^{3c} \log n$-wise independent (with a small bias, of course) and MaxSet contains at most $c \cdot 4^{2c} \log n < 4^{3c} \log n$ bits, the probability that at least one of the bits in the j^{th} block belonging to *some* I_i in MaxSet becomes good is at least $1 - (1 - \frac{1}{4^c})^{4^{2c} \log n} - \frac{1}{n^2} > 1 - \frac{1}{8n}$. If each one of $4n$ blocks has this property, then the probability that each one of them has at least one good bit is at least $\frac{1}{2}$. The same calculation works when we drop from the set MaxSet those I_is that contain a bit from the first $\log m$ unset pairs of bits of length encoder bits (we drop at most $4c \log 2n$ I_is). This is the Case 2: we can keep sufficient number of length encoder bit pairs unset and still have every block having at least one good bit.

Now consider the other possibility: there is a block with $|\text{MaxSet}| < 4^{2c} \log n$. Divide this block into n sub-blocks of equal size as described above. Clearly, one of these sub-blocks will contain *no* bit that belongs to MaxSet, since MaxSet has less than $c \cdot 4^{2c} \log n$ bits. Fix a sub-block that does not intersect with MaxSet. Now if we set all the bits of all the other blocks and sub-blocks and also at most $c \cdot 4^{2c} \log n < 4^{3c} \log n$ length encoder bits (the ones that belong to MaxSet), all the bits in MaxSet would be set and this would mean that each I_i contains at most $c - 1$ unset bits. This is Case 1! □

References

AAI+97. M. Agrawal, E. Allender, R. Impagliazzio, T. Pitassi, and S. Rudich. Reducing the complexity of reductions. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 730–738, 1997.

AAR98. M. Agrawal, E. Allender, and S. Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *J. Comput. Sys. Sci.*, 57:127–143, 1998.

ABI93. E. Allender, J. Balcázar, and N. Immerman. A first-order isomorphism theorem. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, 1993.

AG91. E. Allender and V. Gore. Rudimentary reductions revisited. *Information Processing Letters*, 40:89–95, 1991.

AGHP90. N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 544–553, 1990.

Agr96. M. Agrawal. On the isomorphism problem for weak reducibilities. *J. Comput. Sys. Sci.*, 53(2):267–282, 1996.

Agr01. M. Agrawal. Towards uniform AC^0 isomorphisms. In *Proceedings of the Conference on Computational Complexity*, 2001. to be presented.

BDG88. J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.

BH77. L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 1:305–322, 1977.

BIS90. D. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. Comput. Sys. Sci.*, 74:274–306, 1990.

CSV84. A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13:423–439, 1984.

ER60. P. Erdös and R. Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35:85–90, 1960.

FFK92. S. Fenner, L. Fortnow, and S. Kurtz. The isomorphism conjecture holds relative to an oracle. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 30–39, 1992. To appear in SIAM J. Comput.

FSS84. M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

IL95. N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116:103–116, 1995.

Imm87. N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.

Jon75. N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Sys. Sci.*, 11:68–85, 1975.

KMR88. S. Kurtz, S. Mahaney, and J. Royer. The structure of complete degrees. In A. Selman, editor, *Complexity Theory Retrospective*, pages 108–146. Springer-Verlag, 1988.

KMR89. S. Kurtz, S. Mahaney, and J. Royer. The isomorphism conjecture fails relative to a random oracle. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 157–166, 1989.

Lin92. S. Lindell. A purely logical characterization of circuit complexity. In *Proceedings of the Structure in Complexity Theory Conference*, pages 185–192, 1992.

NN90. J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 213–223, 1990.

NW94. N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. Sys. Sci.*, 49(2):149–167, 1994.

Sip83. M. Sipser. Borel sets and circuit complexity. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 61–69, 1983.

Thresholds and Optimal Binary Comparison Search Trees (Extended Abstract)

Richard Anderson[1], Sampath Kannan[2,⋆], Howard Karloff[3,⋆⋆], and Richard E. Ladner[4,⋆⋆⋆]

[1] Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195.
anderson@cs.washington.edu.
[2] AT&T Labs–Research and Department of CIS, University of Pennsylvania, Philadelphia, PA 19104.
kannan@cis.upenn.edu.
[3] AT&T Labs–Research, Room C231, 180 Park Ave., Florham Park, NJ 07932, and College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., Atlanta, GA 30332-0280.
howard@research.att.com.
[4] Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195.
ladner@cs.washington.edu.

Abstract. We present an $O(n^4)$-time algorithm for the following problem: Given a set of items with known access frequencies, find the optimal binary search tree under the realistic assumption that each comparison can only result in a two-way decision: either an equality comparison or a less-than comparison. This improves the best known result of $O(n^5)$ time, which is based on split tree algorithms. Our algorithm relies on establishing thresholds on the frequency of an item that can occur as an equality comparison at the root of an optimal tree.

1 Introduction

The *binary search tree* (*BST*) is one of the classic data structures in computer science. One of the fundamental problems in this area is how to build an optimal binary search tree where the items stored in the tree have some observed frequencies of access. In addition, there may be failure frequencies for unsuccessful searches. In the traditional problem each node in the binary search tree is labeled with an item a. In a search for an item x, when the node labeled with a is

⋆ Research supported in part by ONR grant N00014-97-1-0505 and NSF grant CCR-9820885.
⋆⋆ Research supported in part by NSF grant CCR-9732746.
⋆⋆⋆ Research performed in part while at AT&T Labs–Research, Florham Park, NJ 07932, and supported in part by NSF grant CCR-9732828.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 83–95, 2001.
© Springer-Verlag Berlin Heidelberg 2001

encountered there are three possible outcomes: $x < a$, $x = a$, and $x > a$. In the first case the search proceeds to the left subtree, in the second case the search ends at the node, and in the third case the search proceeds to the right subtree. A disadvantage of this three-way branching is that it takes two computer instructions to determine the outcome. Knuth [9, Problem 33, page 457] suggests that it would be interesting to explore an alternative binary tree structure where nodes are labeled with either an equality comparison or a less-than comparison. The resulting tree has two-way branching instead of three-way branching. We call such trees *binary comparison search trees* or *BCST*'s for short.

A very simple example demonstrates the benefit of having equality comparisons. Consider the example of three items with weights $(0, 1, 0)$. The optimum with equality has cost 1 while the optimum without equality has cost 2. (We can replace the 0's by arbitrarily small ϵ's to make this example less pathological.)

Several years ago, Spuler [13] exhibited an $O(n^5)$-time algorithm to find the optimal BCST. His algorithm was based on earlier algorithms to find an optimal split tree [5,7,11]. (One node in a split tree [12] has one less-than comparison and one equality comparison, possibly on different items, leading to 3-way branching.)

In this paper we revisit the problem of finding the optimal BCST in the case of just success frequencies. Using a new approach we show that the optimal BCST can be computed in time $O(n^4)$. The algorithm's correctness depends on our result that if all the frequencies are less than one fourth of the sum of the frequencies then the root of an optimal BCST cannot be an equality comparison.

1.1 Practical Motivation

One motivation for our study comes from an increased interest in high performance method dispatching that is required for object-oriented programs. Chambers and Chen [2] describe a *dispatch tree*, with both equality and less-than comparisons, to efficiently look up methods in object-oriented programs. A method is always found so there is no chance of failure. Chambers and Chen employ an interesting and effective heuristic to find a good dispatch tree. They left open the question of how to find an optimal dispatch tree. Dispatch trees are actually slightly more general than our BCST's. We focus on the more restricted problem of finding an optimal BCST.

1.2 Traditional Binary Search Trees

There are several efficient algorithms for finding optimal BST's. A standard $O(n^3)$-time dynamic program can be used, but the time can be improved to $O(n^2)$ by using a clever technique [9].

In a BCST, after a few equality comparisons are done, the resulting subproblem, with its "holes" due to failed equality comparisons, corresponds to anything but an interval $\{i, i+1, i+2, ..., j\}$ of indices. The traditional dynamic-programming algorithm relies on the fact that the only subproblems that arise are intervals $\{i, i+1, i+2, ..., j\}$. *A priori*, with equality comparisons, one

could recursively generate a subproblem corresponding to an arbitrary subset of $\{1, 2, ..., n\}$. We will demonstrate, however, that the number of subproblems needed to find the optimal BCST is $O(n^3)$.

We show that there is always a tree having only less-than comparisons which has cost at most one more than the optimum when both types of comparisons are allowed, if the access frequencies sum to one.

2 Preliminaries

An input is specified by a sorted sequence $(a_1, a_2, \ldots, a_n)$ with corresponding nonnegative weights $(w_1, w_2, \ldots, w_n)$. A solution is a binary comparison search tree (BCST) which is a binary tree with each node labeled with either an *equality comparison* of the form $x = a_i$? or a *less-than comparison* of the form $x < a_i$?. For both types of comparisons the two outcomes are "yes" and "no" and we will assume canonically that the left branch of the node representing the comparison corresponds to the "yes" answer. For any BCST T, there is a bijection between the leaves of the tree and the input items. For a tree T, we let $\mathcal{L}(\mathcal{T})$ denote the leaf set of T.

Definition 1. *The* weight of a node v *(denoted $w(v)$) in a* BCST T *is defined as follows. If v is a leaf then $w(v)$ is the weight of the input item labeling v. Otherwise, the* weight of a node *is the sum of the weights of its children. (In other words, the weight of v is the sum of the weights of all leaves which are descendants of v.) Similarly, define the* weight of a tree *to be the sum of the weights of its leaves.*

Definition 2. *The* cost *of a* BCST T *is defined to be* $c(T) = \sum_{\ell \in \mathcal{L}(T)} w(\ell)\text{depth}(\ell)$, *where the* depth *of a node is the length of the path from the root to that node. Equivalently* $c(T) = \sum_{\ell \in V(T) - \mathcal{L}(T)} w(\ell)$.

An *optimal BCST* is one with minimal cost. We need several more definitions and preliminary lemmas en route to our optimal BCST algorithm.

Definition 3. *The* depth *of a subtree is the depth of its root.*

Definition 4. *The* side-weight *of a node v (denoted $sw(v)$) in a* BCST *is defined as follows. If v is a leaf, then $sw(v) = 0$. If v is a node representing an equality comparison (henceforth referred to as an* equality node*), $sw(v) = w(a_i)$ where a_i is the input item tested for equality at v. If v is a less-than node with children x and y, then $sw(v) = \min\{w(x), w(y)\}$.*

Lemma 1. *Let S be a sequence of items with associated weights and let S_1, S_2 be a "partition" of S into two complementary subsequences. Let T be a* BCST *for S. There there are BCST's T_1 for S_1 and T_2 for S_2 with $c(T_1) + c(T_2) \leq c(T)$.*

Proof. Let T be a BCST for S. Let T_1 be obtained from T by removing all leaves which are in S_2 and repeatedly "shorting out" any node with one child. Clearly T_1 is a BCST for S_1. Similarly obtain BCST T_2 for S_2. It is immediate from the (first) definition of the cost of a BCST that $c(T_1) + c(T_2) \leq c(T)$ and the lemma follows. ■

Note that if T does not have equality comparisons, then neither do T_1 and T_2.

Lemma 2. *Let T be an optimal* BCST. *If u is the parent of v in T, then $sw(u) \geq sw(v)$.*

Proof. Since the side-weight of a leaf is 0, we may assume that neither u nor v is a leaf.

Case 1: Both u, v are less-than comparisons. Assume without loss of generality that v is a right child of u. Let T_1 be the subtree rooted at the child of u which is not v. Let T_2, T_3 be the subtrees rooted, respectively, at the left and right children of v. Let α_i denote the weight of T_i, $1 \leq i \leq 3$. Now $sw(u) = \min\{\alpha_1, \alpha_2 + \alpha_3\}$ and $sw(v) = \min\{\alpha_2, \alpha_3\}$. For a contradiction, assume $sw(u) < sw(v)$, i.e., $\min\{\alpha_1, \alpha_2 + \alpha_3\} < \min\{\alpha_2, \alpha_3\}$, which implies that $\alpha_1 < \alpha_2, \alpha_1 < \alpha_3$. Now rotate v upward along the edge to its parent. While T_2 stays at the same depth, T_1 moves down and T_3 moves up, each by one level. The increase in cost is $\alpha_1 - \alpha_3 < 0$. This contradicts the optimality of T.

Case 2: Node u is a less-than comparison $x < a_1?$ and v is an equality comparison $x = a_2?$. Let T_1, of weight, say, α_1, be the subtree rooted at the child of u which is not v. Let $\alpha_2 = w_2$ and let α_3 be the weight of the subtree T_3 rooted at the child of v not corresponding to a_2. We have $sw(u) = \min\{\alpha_1, \alpha_2 + \alpha_3\}$, $sw(v) = \alpha_2$. For a contradiction, assume that $sw(u) < sw(v)$, i.e., $\min\{\alpha_1, \alpha_2 + \alpha_3\} < \alpha_2$. Hence $\alpha_1 < \alpha_2$.

Again, rotate v up along the edge to u, i.e., replace u by the comparison $x = a_2?$. Replace u's right child by $x < a_1?$. From that node's left and right children, respectively, hang T_1 and T_3. Tree T_1 moves down one level, T_3 stays at the same level, yet a_2, of weight α_2, moves up one level. The net increase in cost is $\alpha_1 - \alpha_2 < 0$, contradicting the optimality of T.

Case 3: Both u, v are equality comparisons, say, u with an item a_1 of weight α_1, v with an item a_2 of weight α_2. We have $sw(u) = \alpha_1, sw(v) = \alpha_2$. For a contradiction, assume $\alpha_1 < \alpha_2$. Swap the comparisons in u and v. Node a_1 moves down, a_2 moves up. The increase in cost is $\alpha_1 - \alpha_2 < 0$, a contradiction.

Case 4: Node u is an equality comparison $x = a_1?$ and v is a less-than comparison $x < a_2?$. Let α_1 be the weight of a_1, and let T_2 and T_3 of weight α_2 and α_3, respectively, be the subtrees hanging off v. Once again assume for a contradiction that $sw(u) < sw(v)$, i.e., $\alpha_1 < \min\{\alpha_2, \alpha_3\}$. Rotate v upward and make the comparison $x = a_1?$ at the appropriate child of v. Then exactly one of T_2 and T_3 moves up while a_1 moves down. The increase in cost is again negative, a contradiction. ■

Corollary 1. *If $n > 2$ and the root of an optimal BCST is an equality node, then the item tested for equality must be a largest-weight item. If $n \leq 2$, then all possible BCST's have the same cost.*

We omit the proof.

Corollary 2. *If there is an item a_m such that $w_m > \frac{1}{2}\sum_{i=1}^{n} w_i$, then there is an optimal BCST having an equality comparison with a_m at the root.*

We omit the proof. We will see later that this factor of 1/2 can be reduced to 4/9.

3 Thresholds

In this section we assume that the sum of the weights is 1 and we refer to the weights as probabilities.

Intuitively, if the maximum probability is large, there should be an optimal BCST whose root comparison is an equality comparison, which, by Corollary 1, must be with an item of maximum probability. Analogously, one would expect that if the maximum probability is small, there should not exist an optimal BCST with a root equality comparison. We study in this section the relationship between the maximum probability and the existence of an optimal BCST with a root equality comparison.

If the maximum is very small, (we will see that) there cannot be an optimal BCST with an equality comparison at the root. Let us define λ to be the supremum of all p such that for any input whose maximum probability is at most p, there is no optimal BCST with an equality comparison at the root. We will prove that if the maximum probability is less than 1/4, then there is no optimal BCST with an equality comparison at the root (hence $\lambda \geq 1/4$), and there is an instance with maximum probability 1/4 which has an optimal BCST with a root equality comparison (hence $\lambda \leq 1/4$). So $\lambda = 1/4$.

How large should the maximum probability be, in order to guarantee the existence of an optimal BCST with a root equality comparison? By Corollary 2, if the maximum probability exceeds 1/2 then there is a BCST in which the root is an equality comparison. Must there be an optimal BCST with an equality comparison in the root if, instead, the maximum probability is, say, 0.4? Let us define μ to be the infimum of all p such that for any input whose maximum probability is at least p, there is an optimal BCST with an equality comparison at the root. We will prove that if the maximum probability is at least 4/9, then there is an optimal BCST having an equality comparison at the root (hence $\mu \leq 4/9$), whereas there are instances with maximum probability approaching 3/7 from below which have no optimal BCST with an equality comparison at the root (hence $\mu \geq 3/7$). So $3/7 \leq \mu \leq 4/9$.

Later, we will use the fact that $\lambda \geq 1/4$ to design a polynomial-time algorithm to find the optimal BCST.

We will use *left* or *right subtree of* T to mean the subtree of T rooted at the left or right child of the root of T.

Theorem 1. $\lambda = 1/4$.

Proof. First we prove that $\lambda \geq 1/4$. Suppose, for a contradiction, that T is an optimal BCST where input item b_0 ($= a_i$ for some i) with weight less than $1/4$ is tested for equality at the root.

Then the side-weight of the root is equal to $w(b_0)$, which is less than $1/4$. By Lemma 2 the side-weight of every other node in the tree is less than $1/4$.

For any internal equality comparison $x = a_i?$ in a node v, let us call the branch leading to the child of v having leaf a_i the *side branch*, and the other branch the *main branch*. For a less-than comparison $x < a_i?$ at node v, we call the branch leading to the child of v of lesser weight the *side branch*, breaking a tie arbitrarily, naming the other branch the *main branch*. The weight of the child along the side branch from v is always $sw(v)$.

Let $r = v_0, v_1, v_2, v_3, ..., v_l$ be the nodes, in order, along the unique path from the root to the leaf v_l along main branches. Let s_i, $0 \leq i \leq l-1$, be the child of v_i on the side branch. Then $sw(v_i) = w(s_i)$. Furthermore, $sw(v_i) \leq sw(v_0) < 1/4$, by Lemma 2. Now $w(T) = [w(s_0) + w(s_1) + w(s_2) + \cdots + w(s_{l-1})] + w(v_l)$. By Corollary 1, $w(v_l) \leq w(s_0) = sw(v_0) < 1/4$. Hence $w(T) \leq (l+1)sw(v_0)$. If $l \leq 3$, then $w(T) \leq 4 \cdot sw(v_0) < 1$, contradicting the fact that $w(T) = 1$. Hence $l \geq 4$.

Now let T_1, T_2, T_3 be the trees hanging off the side branches from v_1, v_2, v_3, respectively, and let T_4 be the tree hanging off the main branch from v_3. Note specifically that $w(b_0) \geq w(T_1) \geq w(T_2) \geq w(T_3)$. Figure 1 illustrates this configuration. In this and other figures in this section, we have departed from the usual convention and chosen to depict the side branches as left children and the main branches as right children, since there may be insufficient information to pin down which branch is the "yes" branch.

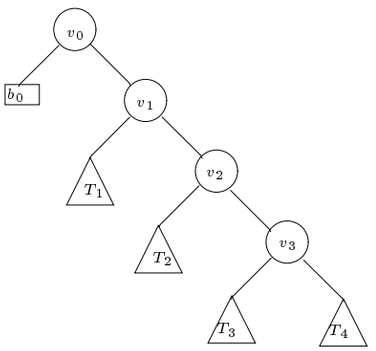


Fig. 1. Tree T expanded along main branches to four levels

A convention will be useful in the proof. Since the identities of the items are actually irrelevant—only the weights matter—we assume that $a_i = i$, $i =$

$1, 2, ..., n$. Now each node v_i along the main branch from the root has either the comparison $x = b_i?$ or the comparison $x < b_i?$, where $b_i \in \{1, 2, ..., n\}$. In either case we will define $b_i \in \{1, 2, 3, ..., n\}$ to be the *cut-point* associated with the comparison at v_i. Pictorially, we will represent the cut-points on a number line with an "x" marking a cut-point corresponding to an equality node and a vertical line marking the cut-point corresponding to a less-than node. Thus our picture has a number line with four cut-points labeled b_0, b_1, b_2, and b_3, corresponding to vertices v_0, v_1, v_2, v_3, respectively.

Note the following fact: If v_i is a less-than node with comparison $x < b_i?$ having cut-point b_i, then for all $j > i$, the cut-points b_j must occur on the same side of b_i, the side which is along the main branch at v_i. In other words, if the main branch corresponds to $x \geq b_i$, the "yes" branch, then $b_j \geq b_i$ for all $j > i$, and if the main branch corresponds to $x < b_i$, then $b_j < b_i$ for all $j > i$. The tree T_i contains only items from the other side of b_i.

Since T_3 and T_4 are symmetrically located in T, we will assume without loss of generality that T_3 contains items to the left of b_3 and T_4 contains b_3 and items to the right.

The idea of the proof is to show that we can rebalance the "skinny" tree shown in Figure 1 (making the root node a less-than), and reduce its cost, thereby contradicting the optimality of this tree. Since there are four cut-points corresponding to the tree in Figure 1 it is natural that the balanced tree T' should have at its root a less-than node which splits these cut-points equally. We will call the less-than comparison at the root of T' the *dividing cut*.

There are two main cases.

1. If the middle two cut-points b_i and b_j of T both correspond to equality nodes, then we will let the dividing cut be $x < b_j?$ where $b_j > b_i$.
2. Otherwise, (at least) one of the middle cut-points corresponds to a less-than comparison $x < b_i?$. In this case, (choose one and) let this less-than comparison be the dividing cut.

Case 1: Note that in this case there are two cut-points to be dealt with on either side of the dividing cut. Let m, p, $0 \leq m < p \leq 3$, be such that the set of two cut-points to the left of b_j is $\{b_m, b_p\}$. Then we perform the comparison occurring in node v_m of T at the left child of the root of T' and perform the comparison occurring in node v_p of T at the appropriate child of this left child. At the other subtree of this left child, put the optimal BCST for the appropriate set of items. We similarly handle the right side of the dividing cut: Where the rightmost two cut-points are b_q, b_r with $q < r$, do the comparison with b_q and then the comparison with b_r. Intuitively, since $m < p$, $w(T_m) \geq w(T_p)$ and we want to have T_m occur higher in T'. This is the reason for the above ordering of comparisons.

Note that in this case, the dividing cut introduces a new split and thereby fractures one of the subtrees in T. Note also that the subtree fractured must be T_3 or T_4. To see this, note that in order for T_i to be fractured, v_i must be a less-than comparison (otherwise T_i consists of a single node) and hence b_i must be one of the two end cut-points. If b_i is the leftmost cut-point and $i < 3$, then

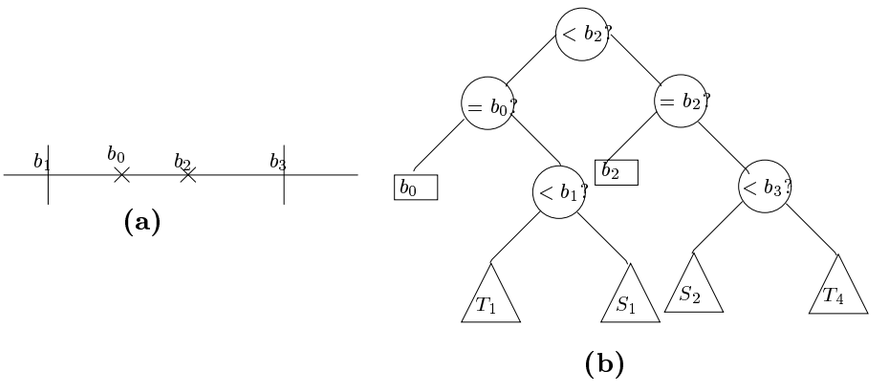


Fig. 2. Example scenario where the middle two cut-points are equalities, and the resulting tree T'

since b_3 occurs to the right of b_i, T_i must be to the left and is not fractured. A similar argument holds if b_i is the rightmost cut-point. Whichever of T_3 or T_4 is fractured, we will let S_1 and S_2 denote the subtrees for the two pieces obtained using Lemma 1. Figure 2 shows one example scenario that falls within this case, and the resulting T'.

We will compare the costs of T and T' by looking at the changes in the depths of (the leaf corresponding to) b_0 and the roots of T_1, T_2, and T_3. The depth of b_0 goes from 1 to 2 since the comparison $x = b_0$? will be done at a child of the root of T'. There are two cases for the change in the depth of T_1. If b_1 occurs on the same side of the dividing cut as b_0, then the depth of the root of T_1 goes from 2 to 3. Otherwise, it remains 2. In the former case, the depth of T_2 goes from 3 to 2, while in the latter case, the depth of T_2 remains 3. Since $w(T_1) \geq w(T_2)$, the worst possible improvement is when the depth of T_1 goes from 2 to 3. Note that the depth of all three pieces that constitute the original T_3 and T_4 goes from 4 to 3.

Using Lemma 1 for the fractured subtree, $c(T') - c(T) \leq w(b_0) + [w(T_1) - w(T_2)] - w(T_3) - w(T_4) = w(b_0) + w(T_1) - (1 - w(b_0) - w(T_1)) = 2(w(b_0) + w(T_1)) - 1$. Since $w(T_1) \leq w(b_0) < 1/4$, the bound above is negative, showing that the cost of T' is less than the cost of T and contradicting the assumption that T is optimal.

Case 2: Since one of the four cut-points of T has already been dealt with as the dividing cut, we will have two cut-points to take care of on one side of the dividing cut and only one to take care of on the other. Just as in Case 1, the two cut-points on one side of the dividing cut are sequenced in the same order in T' as in T. The subtrees T_i, $i = 1, \ldots, 4$, will not be fractured in this case since the cuts we will use in T' are the same as the cuts used in T.

Consider first the situation where the dividing cut corresponds to cut-point b_1. Since b_2 and b_3 are on the same side of b_1, b_0 must be the lone cut-point

on one side and b_2 and b_3 must be on the other side. In the resulting tree T', the depth of b_0 goes from 1 to 2, the depth of T_1 stays at 2, the depth of T_2 goes from 3 to 2 and the depths of T_3 and T_4 go from 4 to 3. Thus $c(T') - c(T) \leq w(b_0) - w(T_2) - w(T_3) - w(T_4) < 0$, a contradiction.

Next consider the case where the dividing cut corresponds to cut-point b_2. An example of this situation when b_3 is the lone cut-point on one side of the dividing cut is shown in Figure 3. In this case, the depths of b_0 and T_1 increase by 1, the depth of T_2 is unchanged and the depths of T_3 and T_4 decrease by 2. Since $w(T_3) + w(T_4)$ is at least $1/4$ and $w(b_0) + w(T_1)$ is less than $1/2$, the net change in cost is negative, again contradicting the optimality of T. If the lone cut-point on one side of the dividing cut is either b_0 or b_1, then in the resulting T' the depth of b_0 increases by 1, the depth of T_1 is unchanged and the depths of T_2, T_3 and T_4 decrease by 1, again yielding a net reduction in cost.

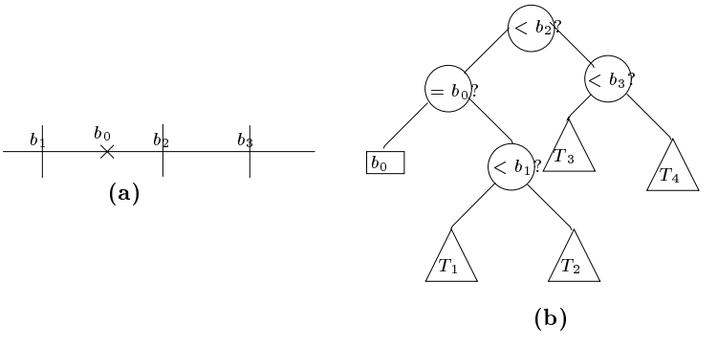


Fig. 3. Example scenario where b_2 is the dividing cut, and the resulting tree T'

Finally consider the situation where the dividing cut corresponds to cut-point b_3. If b_2 is the sole cut on one side of the dividing cut, then the depths of b_0 and T_1 increase by 1, the depth of T_2 drops by 1 and the depths of T_3 and T_4 drop by at least 1, giving a net reduction in cost. If b_0 or b_1 is the sole cut-point on one side of the dividing cut, then the depth of b_0 increases by 1, the depths of T_1 and T_2 are unchanged and the depths of T_3 and T_4 drop by at least 1, again giving us the result. Figure 4 depicts one case of this situation.

To prove that $\lambda \leq 1/4$, we consider a six-item example $(a_1, a_2, \ldots, a_6)$ with weights $(1/4, 0, 1/4, 1/4, 0, 1/4)$. By a case analysis, an optimal BCST has cost $5/2$ and at least one optimal tree has root equality comparison $x = a_3$?. ■

Theorem 2. $\mu \leq 4/9$.

We omit the (long) proof.

Theorem 3. $\mu \geq 3/7$.

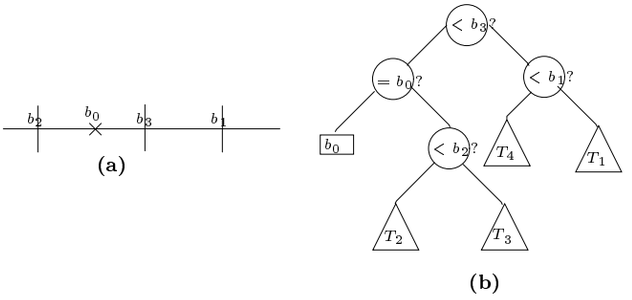


Fig. 4. Example scenario where b_3 is the dividing cut, and the resulting tree T'

Proof. Consider the weights $(3/7-4\epsilon, 1/7+\epsilon, 0, 1/7+\epsilon, 1/7+\epsilon, 0, 1/7+\epsilon)$ for a 7-item example $(a_1, a_2, \ldots, a_7)$. By a case analysis, it can be shown that the *unique* optimal BCST, with cost $17/7 + 3\epsilon$ for all sufficiently small positive ϵ, has root comparison $x < a_3$?. By contrast the lowest weight tree with root comparison $x = a_1$? has weight $17/7 + 10\epsilon$. ■

4 The $O(n^4)$-Time Algorithm

In this section we give an $O(n^4)$-time, dynamic programming-based algorithm for finding an optimal BCST on n items. The algorithm relies heavily on the fact that the initial comparison cannot be $x = a_i$? unless a_i has the maximum weight and that weight is at least 1/4 of the sum of all weights. Since the identities of the items $a_1, a_2, ..., a_n$ are irrelevant, the input consists of a sequence $< w_1, w_2, ..., w_n >$ of nonnegative weights, assumed integral in this section.

Our algorithm will compute the optimal cost for each of at most $16n^3$ subproblems, computing each one in $O(n)$ time. The notation $S =< i_1, i_2, i_3, ..., i_r >$ with $1 \leq i_1 < i_2 < \cdots < i_r \leq n$ or $S = \{i_1, i_2, i_3, ..., i_r\}$ denotes the subproblem of finding the optimal BCST for the items numbered $i_1, i_2, i_3, ..., i_r$, with associated respective weights $w_{i_1}, w_{i_2}, w_{i_3}, ..., w_{i_r}$.

We will compute the optimal cost of each "valid" subproblem:

Definition 5. *Let S be a nonempty subset of $\{1, 2, ..., n\}$. Let $i = \min S$, $j = \max S$ (possibly $i = j$), and $M = \max_{l \in S} w_l$. S is* valid *if and only if it satisfies these two conditions: (1) S is closed (strictly) downward in the interval $[i, j]$ in that S contains every $l \in [i, j]$ such that $w_l < M$, and (2) For $T = \{l | i \leq l \leq j : w_l = M\}$, either $|T| \leq 4$ or $T \subseteq S$.*

First, a simple lemma.

Lemma 3. *The number of valid sets is at most $16n^3$.*

We omit the proof.

We will compute the optimal cost of valid sets in order of increasing size of valid sets, starting with valid sets of size one. To do so, we need a simple

name for each valid set. Give valid set S a unique name (i, j, k, v) as follows. Let $i = \min S$, $j = \max S$. Let $M = \max_{l \in S} w_l$ and let $k = \min\{l | l \in S, w_l = M\}$ (the leftmost position of an M in S). Let $i \leq j_1 < j_2 < \cdots < j_d \leq j$ be all the positions in which M appears as a weight of an item in $[i, j]$ in the original list. Then $v = *$ if $d \geq 5$, in which case S contains all the positions in $[i, j]$ corresponding to weight M. Otherwise, $v \in \{0, 1\}^d$ with $v_t = 1$ iff $j_t \in S$. It is easy in $O(n)$ time to convert from the name of a valid set to an enumeration of its items or *vice versa*.

The first step of the algorithm is to enumerate all valid sets, via their names, to calculate the size of each. We prepare, for each $w \in \{1, 2, ..., n\}$, a list of the names of all valid sets of size w. We prepare an array $cost(i, j, k, v)$, to contain, at termination, the optimal cost of the corresponding valid set. This takes $O(n^4)$ time. The solution for the valid set $\{1, 2, 3, ..., n\}$ is the optimal cost.

We initialize $cost(i, j, k, v)$ to 0 for all valid sets of size 1, and otherwise we initialize $cost(i, j, k, v)$ to $+\infty$. Then in increasing order by w, we calculate the optimal cost of each valid set S of size w as follows. In $O(n)$ time, we enumerate the items $< i_1, i_2, ..., i_w >$ of S in increasing order and the weight W of S.

We consider first the possibility that the root comparison in some optimal BCST for S is a less-than comparison. To do so, we must consider the subproblems $< i_1 >$ and $< i_2, i_3, i_4, ..., i_w >$ (associated with $x < a_{i_2}?$), $< i_1, i_2 >$ and $< i_3, i_4, i_5, ..., i_w >$ (associated with $x < a_{i_3}?$), ..., and $< i_1, i_2, i_3, ..., i_{w-1} >$ and $< i_w >$ (associated with $x < a_{i_w}?$). The key point is that each of these subproblems is valid—see Lemma 4—and smaller than S, so we already know the optimal cost of each. Furthermore, in $O(n)$ time in total, one can construct the names of *all* of them (Lemma 6). For each $t = 1, 2, 3, ..., w - 1$, let C_1 denote the cost of the left subproblem $\{i_1, i_2, ..., i_t\}$ and let C_2 be the cost of the right subproblem $\{i_{t+1}, i_{t+2}, ..., i_w\}$. We now replace $cost(i, j, k, v)$ by the cost $W + C_1 + C_2$ of the optimal tree rooted at $x < a_{i_{t+1}}?$, if it is smaller than $cost(i, j, k, v)$.

Now we deal with the possibility that the root comparison in problem (i, j, k, v) is an equality comparison. We use the index k to find the largest weight M in S. In $O(n)$ time, we find all occurrences of M in S and simultaneously sum the weights corresponding to positions in S. If M is less than a quarter of the sum, then we know that the root comparison cannot be an equality comparison, so we move on to the next valid set. Otherwise, M occurs at most four times in S. For each of the at most four indices i_t with $w_{i_t} = M$ in S, generate the subproblem $S - \{i_t\}$, which is valid (Lemma 5), generate its name, look up its optimal cost C, and replace $cost(i, j, k, v)$ by $W + C$ if it is smaller. All of this can be done in $O(n)$ time for each t.

It is clear that the algorithm runs in $O(n^4)$ time. Furthermore, assuming that *cost* is correct for smaller sets, the construction of the algorithm ensures that the final value of $cost(i, j, k, v)$ is an upper bound on the true optimal cost. The fact that the optimal tree must begin either with (1), a less-than comparison, or (2), an equality comparison on an item which is simultaneously the max and of

at least one fourth the total weight, ensures that $cost(i, j, k, v)$ is a lower bound. Hence the algorithm is correct.

Lemma 4. *Suppose* $S =< i_1, i_2, ..., i_w >$ *is valid. Then* $S' =< i_1, i_2, ..., i_t >$ *and* $S'' =< i_{t+1}, i_{t+2}, ..., i_w >$ *are valid for all* t, $1 \leq t \leq w - 1$.

We omit the proof.

Lemma 5. *Suppose* $S =< i_1, ..., i_w >$ *is valid,* $|S| \geq 2$, $M = \max\{w_{i_j} | 1 \leq j \leq w\}$, $|\{l | i_1 \leq l \leq i_w : w_l = M\}| \leq 4$, *and* $1 < s < w$ *is such that* $w_{i_s} = M$. *Then* $S' = S - \{i_s\}$ *is valid.*

We omit the proof.

Lemma 6. *There is an* $O(n)$*-time algorithm that takes a valid set* $< i_1, i_2, ..., i_w >$ *with* $1 \leq i_1 < i_2 < \cdots < i_w \leq n$ *as input and calculates the names of the subproblems* $< i_1, i_2, ..., i_t >$ *and* $< i_{t+1}, i_{t+2}, ..., i_w >$ *for all* t.

We omit the proof, which uses running prefix and suffix computations.

5 Comparison with Other Models

5.1 BCST's With Only Less-Than Comparisons

The 3-item example with weights $(0, 1, 0)$ has optimal cost 1 when equality and less-than comparisons are allowed but optimal cost 2 when only less-than comparisons are allowed. The following theorem demonstrates that this is the worst possible case.

Theorem 4. *If* T *is a BCST in which the weights sum to 1, then there is a BCST* T' *that uses only less-than comparisons such that* $c(T') \leq c(T) + 1$.

We omit the proof.

References

1. P.J. Bayer. Improved Bounds on the Cost of Optimal and Balanced Binary Search Trees. M.Sc. Thesis, MIT, MIT/LCS/TM-69, 1975.
2. C. Chambers and W. Chen. Efficient Multiple and Predicate Dispatching. Proceedings of the 1999 ACM Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA '99), November, 1999.
3. R.G. Gallager. *Information Theory and Reliable Communication.* Wiley, New York, 1968.
4. A.M. Garsia and M.L. Wachs. A New Algorithm for Minimum Cost Binary Trees. *SIAM Journal on Computing*, Vol. 6, pp. 622-242, 1977.
5. J.H. Hester, D.S. Hirschberg, S.-H.S. Huang, and C.K. Wong. Faster Construction of Optimal Binary Split Trees. *Journal of Algorithms*, Vol. 7, pp. 412-424, 1986.

6. T.C. Hu and A.C. Tucker. Optimal Computer-Search Trees, and Variable Length Alphabetic Codes. *SIAM Journal on Applied Mathematics*, Vol. 21, pp. 514-532, 1971.
7. S.-H.S. Huang and C.K. Wong. Optimal Binary Split Trees. *Journal of Algorithms*, Vol. 5, pp. 69-79, 1984.
8. D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. Proc. Institute of Radio Engineers, Vol. 40, pp. 1098-1101, 1952.
9. D.E. Knuth. *The Art of Computer Programming: Volume 3, Second Edition, Sorting and Searching.* Addison-Wesley, Reading, Massachusetts, 1998.
10. K. Mehlhorn. A Best Possible Bound for the Weighted Path Length of Binary Search Trees. *SIAM Journal on Computing*, Vol. 6, pp. 235-239, 1977.
11. Y. Perl. Optimum Split Trees. *Journal of Algorithms*, Vol. 5, pp. 367-374, 1984.
12. B.A. Sheil. Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys. *Communications of the ACM*, Vol. 21, pp. 947-958, 1978.
13. D. Spuler. Optimal Search Trees Using Two-Way Key Comparisons. *Acta Informatica*, Vol. 32, pp. 729-740, 1994.

Distributed LTL Model Checking Based on Negative Cycle Detection*

Luboš Brim, Ivana Černá, Pavel Krčál, and Radek Pelánek

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
{brim,cerna,xkrcal,xpelanek}@fi.muni.cz

Abstract. This paper addresses the state explosion problem in automata based LTL model checking. To deal with large space requirements we turn to use a distributed approach. All the known methods for automata based model checking are based on depth first traversal of the state space which is difficult to parallelise as the ordering in which vertices are visited plays an important role. We come up with entirely different approach which is dependent on locating cycles with negative length in a directed graph with real number length of edges. Our method allows reasonable distribution and the experimental results confirm its usefulness for distributed model checking.

1 Introduction

Model checking is a very successful technique for verifying concurrent systems and many verification tools were proposed in the last two decades. These tools verify a desired behavioural property of a reactive system over a given model through exhaustive enumeration of all the states reachable by the system and the behaviours that traverse through them. As a matter of fact, the main limiting factor in applications of such tools to practical verification problems is the real computational power available (time and especially memory). Therefore verification of complex concurrent systems requires techniques to avoid the state-explosion problem [9]. Several sequential methods (partial order reductions, on-the-fly search) to overcome this barrier have been proposed and successfully implemented in automatic verification tools. Recently, some attempts to use multiprocessors and networks of workstations have been undertaken.

In [23] the authors describe a parallel version of the verifier Murφ. The table of all reached states is partitioned over the nodes of the parallel machine and the explicit state enumeration is performed in parallel. A similar approach to distributed reachability analysis has been taken in [18]. A distributed version of the UPPAAL model checker based on the same idea as parallel Murφ has been reported in [3]. Yet another distributed reachability algorithm has been proposed in [1], but has not been implemented. We stress that all mentioned

* This work has been partially supported by the Grant Agency of Czech Republic grants No. 201/00/1023 and 201/00/0400.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 96–107, 2001.
© Springer-Verlag Berlin Heidelberg 2001

algorithms solve only the reachability problem and do not admit the complete linear time model checking. A distributed version of the LTL model checker SPIN [16] based on nested depth first search approach has been explored in [2]. Other recent papers attempt to use distributed environment of workstations for parallel symbolic model checking. [15] presents a parallel reachability analysis algorithm based on BDDs while in [4] distributed symbolic method has been applied to check safety RCTL properties. Papers [14,5] significantly extend the scope of properties that can be verified by presenting distributed symbolic model checking for μ-calculus and alternation free μ-calculus.

In automata based LTL model checking the verification problem is represented as the emptiness problem of a Büchi automaton which turns out to be equivalent to finding a cycle reachable from an initial state and containing an accepting state in the graph corresponding to the Büchi automaton. The best known algorithm for finding cycles in directed graphs is the Tarjan's depth first search algorithm (DFS) [24]. The practical limitation of this algorithm is the amount of the randomly accessed memory which the algorithm requires. A space efficient alternative to Tarjan's algorithm (so called nested DFS) allowing to optimise the amount of randomly accessed memory exists (see i.e. [17]) and is implemented in SPIN verification tool [16]. However, even this optimisation does not solve the state space explosion problem sufficiently.

A very natural way how to overcome the memory limitation is to distribute the given graph onto several processors (computers) and to perform a distributed computation. As depth first search is P-complete, promising parallel DFS-based algorithms are unlikely to exist [21]. A completely different approach to distributed emptiness problem is needed. This paper demonstrates the methodology of reducing the automata based LTL model checking problem to the *negative cycle detection* problem. The problem is to find a negative length cycle in a directed graph whose edges have real number lengths.

The problem of negative cycles is closely related to the single-source shortest path (SSSP) problem. For this problem effective PRAM algorithms working with adjacency matrix representation of graphs are known, see i.e. [22]. However, the adjacency matrix representation is not compatible with other space-saving techniques like *on-the-fly* search. Other algorithms (for excellent survey see [8]), which are based on relaxation of graph's edges, are inherently sequential and their parallel versions are known only for special settings of the problem. For general digraphs with non-negative edge lengths parallel algorithms are presented in [19,20,12]. For special cases of graphs, like planar digraphs [25,13], graphs with separator decomposition [10] or graphs with small tree-width [7] more efficient algorithms are known. Yet none of these algorithms is applicable on directed graphs with potential negative cycles.

We present a scalable distributed algorithm for the negative cycle problem and thus for automata based model checking of LTL formulas. Our method parallelises the model checking problem on a network of processors with disjoint memory that communicate via message passing.

The paper is organised as follows. We first review automata based LTL model checking and define the corresponding graph theoretic problem (Section 2). Its reduction to the negative cycle problem is outlined in Section 3. A distributed algorithm for the negative cycle problem is given in Section 4. Section 5 summarises the experimental results achieved.

2 Automata Based LTL Model Checking

Automata based approach to model checking of linear temporal logic formulas is a very elegant method developed by Vardi and Wolper [26]. The essence of using automata for model checking is that both the modelled system and the specification the system is supposed to fulfil are represented in the same way — as Büchi automata.

Definition 1. A Büchi automaton *is a tuple* $A = (\Sigma, S, s, \rho, F)$*, where*

- Σ *is a finite alphabet*
- S *is a finite set of states*
- $s \in S$ *is the initial state*
- $\rho : S \times \Sigma \to 2^S$ *is a transition relation*
- $F \subseteq S$ *is a set of accepting states*

A run *of A over an infinite word $w = a_1 a_2 \ldots$ is a sequence $s_0, s_1, \ldots$ such that for all $i \geq 1 : s_i \in \rho(s_{i-1}, a_i)$. A run $s_0, s_1, \ldots$ over w is* accepting *iff $s_0 = s$ and $\{t \mid t = s_i$ infinitely often$\} \cap F \neq \emptyset$. A word w is* accepted *by A if there is an accepting run over w. The set of words accepted by A is denoted by $L(A)$.*

States of the modelled finite-state system M are identified with the states of a Büchi automaton A_M where all the states are accepting. Then, the set of behaviours of the system is the language $L(A_M)$. On the other hand, for each LTL formula φ one can construct a Büchi automaton A_φ that accepts exactly the set of runs satisfying the formula φ. Hence for the system M and LTL formula φ the verification problem is to verify whether $L(A_M) \subseteq L(A_\varphi)$ or equivalently whether $L(A_M) \cap L(A_{\neg\varphi})$ is empty. Moreover one can build an automaton A for $L(A_M) \cap L(A_{\neg\varphi})$ having $\mid M \mid 2^{\mathcal{O}(\mid\varphi\mid)}$ states. We need to check this automaton for emptiness [26].

Let $A = (\Sigma, S, s, \rho, F)$ be a given automaton. Consider the directed graph $G_A = (S, E_A)$ such that $E_A = \{(u, v) \mid v \in \rho(u, a), a \in \Sigma\}$. The following assertion can be easily verified [26].

Theorem 1. *Let A be a Büchi automaton. Then $L(A)$ is non-empty iff G_A has a cycle that is reachable from the initial state s and contains some accepting state.*

Detection of a reachable accepting cycle in a graph corresponding to a Büchi automaton is thus at the heart of most automata based model checkers. The depth first search strategy (DFS) provides a suitable time efficient approach. However, in large applications graphs are often too massive to fit completely

inside the computer's internal memory. The resulting input/output paging between fast internal memory and slower external memory (such as disks) is then a major performance bottleneck.

In order to overcome problems with the limited size of randomly accessed memory we suggest to divide the graph onto several processors. The simplest solution is to run some DFS based algorithm on those processors. Instead of paging, computation is handed over to a processor owning related data i.e. paging is substituted by communication. As communication among processors is rather time consuming this approach could end up with algorithms which are comparatively slow (this finding is supported by experiments presented in Section 5).

Our methodology is based on the reduction of the Büchi automaton emptiness problem to a problem of detecting a negative cycle in an directed graph as is illustrated in the following section.

3 Negative Cycles

The negative cycle problem is a well-studied problem in connection with the single-source shortest path (SSSP) problem. We are given a triple (G, s, l), where $G = (V, E)$ is a directed graph with n vertices and m edges, $l : E \to R$ is a *length function* mapping edges to real-valued lengths, and $s \in V$ is the *source* vertex. The *length of path* $\rho =< v_0, v_1, \ldots, v_k >$ is the sum of the lengths of its constituent edges, $l(\rho) = \sum_{i=1}^{k} l(v_{i-1}, v_i)$. We define the *shortest path length* from s to v by $\delta(s, v) = \min\{l(\rho) \mid \rho \text{ is a path from } s \text{ to } v\}$ if there is such a path and $\delta(s, v) = \infty$ otherwise. A *shortest path* from vertex s to vertex v is then defined as any path ρ with length $l(\rho) = \delta(s, v)$. If the graph G contains no cycle c with negative length $l(c)$ (*negative cycle*) that is reachable from source vertex s, then for all $v \in V$ the shortest path length remains well-defined and the graph is called *feasible*. If there is a negative cycle reachable from s, shortest paths are not well-defined as no path from s to a vertex on the cycle can be a shortest path. If there is a negative cycle on some path from s to v, we define $\delta(s, v) = -\infty$.

The SSSP problem is to decide whether, for a given triple (G, s, l), the graph G is feasible and if it is then to compute shortest paths from the source vertex s to all vertices $v \in V$. The negative cycle problem is to decide whether G is feasible.

The connection between the negative cycle problem and the Büchi automaton emptiness problem is the following. A Büchi automaton corresponds to a directed graph G_A as defined in Section 2. Let us assign lengths to its edges in such a way that all edges out-coming from vertices corresponding to accepting states have length -1 and all others have length 0. With this length assignment, negative cycles simply coincide with accepting cycles and the problem of Büchi automaton emptiness reduces to the negative cycle problem.

Theorem 2. *Let A be a Büchi automaton. Let $G^A = (G_A, s, l)$ where $l : E_A \to \{0, -1\}$ is the length function such that $l(u, v) = -1$ iff $u \in F$. Then $L(A)$ is non-empty iff G^A has a negative cycle reachable from s.*

4 Distributed Negative Cycle Detection Algorithm

The general sequential method for solving the SSSP problem is the *scanning* method [11,8]. For every vertex v, the method maintains its distance label $d(v)$, parent vertex $p(v)$ and status $S(v) \in \{unreached, labelled, scanned\}$. The subgraph G_p of G induced by edges $(p(v), v)$ for all v such that $p(v) \neq nil$, is called the *parent graph*. Initially for every vertex v, $d(v) = \infty$, $p(v) = nil$ and $S(v) = unreached$. The method starts by setting $d(s) = 0$, $p(s) = nil$ and $S(s) = labelled$. At every step, the method selects a *labelled* vertex v and applies to it a scanning operation. During scanning a vertex v, every edge (v, u) outcoming from v is *relaxed* which means that if $d(u) > d(v) + l(v, u)$ then $d(u)$ is set to $d(v) + l(v, u)$ and $p(u)$ is set to v. The status of v is changed to *scanned* while the status of u is changed to *labelled*. If all vertices are either *scanned* or *unreached* then d gives the shortest path lengths and G_p is the graph of shortest paths.

Different strategies for selecting a *labelled* vertex to be scanned next lead to different algorithms. Our strategy comes out from the Bellman-Ford-Moore [8] algorithm which uses FIFO strategy to select a labelled vertex. The next vertex to be scanned is removed from the head of the queue; a vertex that becomes labelled is added to the tail of the queue if it is not already on the queue.

For graphs where negative cycles could exit the scanning method must be modified to recognise the unfeasibility of the graph. As in the case of scanning various strategies are used to detect negative cycles [8]. However, not all of them are suitable for our purposes – they are either uncompetitive (as for example time-out strategy) or they are not suitable for distribution (such as the admissible graph search which uses hardly parallelizable DFS or the level-based strategy which employs global data structures). For our distributed algorithm we have used the *walk to root* strategy.

The *walk to root* strategy is based on the fact that any cycle in G_p is a negative cycle. Suppose the relaxation operation applies to an edge (v, u) (i.e. $d(u) > d(v) + l(v, u)$) and the parent graph G_p is acyclic. This operation creates a cycle in G_p if and only if u is an ancestor of v in the current tree. This can be detected by following the parent pointers from v to s. If the vertex u lies on this path then there is a negative cycle; otherwise the relaxation operation does not create a cycle. However, the *walk to root* method increases the cost of applying the relaxation operation to an edge to $\mathcal{O}(n)$ since the cost of the search is $\mathcal{O}(n)$. Therefore the *walk to root* is performed only after the underlying relaxation algorithm performs $\Omega(n)$ work. The running time of *walk to root* is thus amortised over the relaxation time and overall time complexity is increased only by a constant factor. To preserve the termination of the strategy we will change and explain its behaviour afterwards.

The negative cycle detection algorithm *NC* we are proposing works in a distributed environment (no global information is directly accessible) where all processors communicate via message passing. We suppose that the set of vertices of the inspected graph is divided into disjoint subsets. The distribution is determined by the function *owner* which assigns every vertex v to a processor α.

For every vertex v processor $owner(v)$ knows its adjacency list. The distribution can be realized *on-the-fly*. Each processor α is responsible for its own part $G^\alpha = (V_\alpha, E_\alpha)$ of the graph G determined by the owned subset of vertices. Good partition of vertices among processors is important because it has direct impact on communication complexity and thus on run-time of the program. We do not discuss it here because it is itself quite a difficult problem and depends on the concrete application.

The main idea of the distributed algorithm *NC* can be summarised as follows. The distributed computation is initiated by the process *Manager* which performs the necessary initialisations. All processors participating in the algorithm execute the same program. Each processor performs repeatedly the basic scanning operation on all its vertices with *labelled* status (procedure *MAIN*). Such vertices are maintained in the processor's local *queue* Q^α. To process a vertex v which belongs to a different processor a message is sent to the owner of v. In each iteration it first processes messages received from other processors. Several types of messages could arrive:

- a request to update parameters of a vertex u. The procedure *UPDATE* compares the current value $d(u)$ with the received one. If needed, parameters are updated and the vertex u is placed into the *queue*.
- a request to continue in a walk, satisfied by executing the *WTR* procedure.
- a request to continue in removing marks, satisfied by executing the *REM* procedure.

Pseudo-Code of the Distributed Algorithm *NC*

```
1  proc MAIN() {running on each processor α}
2    stamp := 0;
3    if α = Manager then Q^α = {s}; d(s) := 0; p(s) := nil else Q^α := ∅ fi
4    while not finished do process_messages; v := pop(Q^α); SCAN(v) od
5  end
```

```
1  proc SCAN(v)
2    foreach (v,u) ∈ E do
3      if owner(u) = α
4        then UPDATE(u, v, d(v) + l(v,u))
5         else send_message(owner(u), "start UPDATE(u, v, d(v) + l(v,u))") fi od
6  end
```

```
1  proc UPDATE(u, v, t)
2    if d(u) > t then if walk(u) ≠ [nil, nil]
3                       then if owner(v) = α
4                              then push(Q^α, v)
5                               else send_message(owner(v), "do push(Q, v)") fi
6                       else d(u) := t; p(u) := v;
7                            if WTR_amortization then WTR([u, stamp], u);
8                                                    stamp++ fi;
9                            if u ∉ Q^α then push(Q^α, u) fi fi fi
10 end
```

```
1  proc WTR([origin, stamp], at) {Walk To Root}
2    done := false;
3    while ¬done do
4          if owner(at) = α
5            then
6                 if walk(at) = [origin, stamp] →
7                   send_message(Manager, "negative cycle found");
8                   terminate
9                 □ (at = source) ∨ (walk(at) > [origin, stamp]) →
10                  if origin ∈ V_α
11                    then REM([origin, stamp], origin)
12                     else send_message(owner(origin),
13                                  "start REM([origin, stamp], origin))" fi
14                  done := true;
15                □ (walk(at) = [nil, nil]) ∨ (walk(at) < [origin, stamp]) →
16                  walk(at) := [origin, stamp];
17                  at := p(at)
18                fi
19           else
20                send_message(owner(at), "start WTR([origin, stamp], at)");
21                done := true
22         fi
23   od
24 end
```

```
1  proc REM([origin, stamp], at) {Remove Marks}
2    done := false;
3    while ¬done do
4          if owner(at) = α
5            then if walk(at) = [origin, stamp]
6                   then walk(at) := [nil, nil];
7                        at := p(at)
8                    else
9                        done := true fi
10             else send_message(owner(at), start REM([origin, stamp], at));
11                  done := true fi
12   od
13 end
```

The *SCAN* procedure scans a vertex v. Every edge (v, u) outcoming from v is relaxed which means that if $d(u) > d(v) + l(v, u)$ then $d(u)$ is set to $d(v) + l(v, u)$ and $p(u)$ is set to v. If the vertex u lies on a walk to root path its parameters are not changed and the vertex v is placed back into the queue.

The *WTR* procedure is responsible for the negative cycle detection. The procedure follows the parent pointers starting from the state where the procedure has been invoked (*origin*). It is initiated after relaxation of an edge and according to a suitable amortisation strategy (*WTR_amortisation* condition becomes true every n-th time it is called). In the distributed environment it may be the case that even if the vertex v does not lie on any cycle, the parent graph can contain a cycle created in the meantime by some other processor. It can happen that

WTR initiated from v reaches such a cycle and never finishes. The amortisation brings about this problem as well. To fix it each processor maintains a *counter* of started *WTR* procedures. *WTR* marks (variable *walk*) each vertex through which it proceeds by the name of the vertex where the walk has been initiated (*origin*) and the current value of the processor counter (*stamp*). A cycle is detected whenever a vertex with the actual *origin* and *stamp* is reached.

Moreover, it can happen that more than one *WTR* procedure is active at a time. In such a situation the concurrent walks could overwrite its own marks preventing thus detection of a cycle. It is sufficient to complete only one of them – if there is a cycle it will be detected. To decide which walk should continue let us suppose that a total linear ordering on vertices is given. A walk with lower *origin* is stopped.
There are four possible situations that can happen during the walk:

- the procedure reaches the source vertex s (line 9). A negative cycle has not been detected and the *REM* procedure is started.
- the procedure reaches a vertex marked with the same *origin* and the same *stamp* (line 6). This indicates that a negative cycle has been recognised. The cycle can be easily reconstructed by following parent edges. If necessary, the path connecting the cycle with the source vertex can be found using a suitable reachability algorithm.
- the procedure reaches a non-marked vertex, a vertex already marked with lower *origin* or a vertex marked with the same *origin* but lower *stamp* (line 15). The vertex is marked with $[origin, stamp]$ and the walk follows the parent edge.
- the procedure reaches a vertex already marked with higher *origin* (line 9). The walk is stopped and the *REM* procedure is started.

Whenever *WTR* has to continue in a non-local vertex a request to the vertex owner is sent and the local walk is finished.

The purpose of the *REM* procedure is to remove marks introduced by the *WTR* procedure. These marks could otherwise obstruct some possible future runs of *WTR* through marked vertices. Marks to be removed are found with the help of parent edges (this is why the updating of a marked vertex is postponed (line 2 of *UPDATE*)). The *REM* procedure follows the path in the parent graph starting from the *origin* in a similar way as *WTR* does. It finishes when it reaches a source vertex or a vertex marked with different *origin*. However, this does not guarantee that all marks are removed at that very moment. Note that these marks will be removed by some other *REM* procedure eventually. The correctness of cycle detection is guaranteed as for the cycle detection the equality of both *origin* and *stamp* is required.

The distributed algorithm terminates when either all queues of all processors are empty and there are no pending messages or when a negative cycle has been detected. The *Manager* process is used to detect termination and to finish the algorithm by sending a *termination* signal to all the processors.

Theorem 3 (Correctness and Complexity).
If G has no negative cycle reachable from the source s, then the algorithm termi-

nates, $d(v) = \delta(s, v)$ for all vertices $v \in V$, and the parent graph G_p is a shortest path tree rooted at s. Otherwise the existence of a negative cycle is reported. If G is distributed over P processors each of which owns $\mathcal{O}(n/P)$ vertices, then the worst case computation complexity is $\mathcal{O}(n^3/P)$.

For detailed proof of the correctness and the complexity analysis see [6].

5 Experiments

We have implemented the algorithm proposed in Section 4. The implementation has been done in C++ and the experiments have been performed on a cluster of eight 366 MHz Pentium PC Linux workstations with 128 Mbytes of RAM each interconnected with a fast 100Mbps Ethernet and using Message Passing Interface (MPI) library.

In the implementation of the *NC* algorithm we have employed the following optimisation scheme. For more efficient communication between processors we do not send separate messages. The messages are sent in packets of pre-specified size. The optimal size of a packet depends on the network connection and the underlying communication structure. In our case we have achieved the best results for packets of size about 100 single messages.

As far as we know there is no other distributed algorithm for negative cycle problem (see Section 1). Therefore our objective was to compare the performance of the *NC* algorithm with algorithms used in LTL model checkers. For comparison we have used very effective nested depth first search (*NDFS*) algorithm [17] used in SPIN verification tool [16]. In its distributed version the graph is divided over processors like in the *NC* algorithm. Only one processor, namely the one owning the actual vertex in the *NDFS* search, is executing the nested search at a time. The network is in fact running the sequential algorithm with extended memory. The worst case space complexity of *NDFS* is asymptotically the same as the one of our algorithm *NC*. The worst case time complexity of *NDFS* is linear in the number of vertices and edges.

We performed several sets of tests on different instances in order to verify how fast is the algorithm in practice, i.e. beyond its theoretical characterisation. Our experiments were performed on two kinds of systems given by random graphs and generated graphs. Graphs were generated using a simple specification language and an LTL formula. In both cases we tested graphs with and without cycles to model faulty and correct behaviour of systems. As our real example we tested the parametrised Dining Philosophers problem. Each instance is characterised by the number of vertices and the number of cross-edges. The number of cross-edges significantly influences the overall performance of distributed algorithms.

For each experiment we report the average time in minutes and the number of sent messages (communication) as the main metrics. Table 1 summarises the achieved results.

The experiments lead basically to the following conclusions:

- *NC* algorithm is comparable with the *NDFS* one on all graphs.
- *NC* algorithm is significantly better on graphs without negative cycles.

Table 1. Summary of experimental results

			NDFS		NC	
Vertices		Cross–edges	Time	Messages	Time	Messages
Generated, without cycle						
	40398	34854	1:01	79376	0:11	809
	71040	1301094	31:13	3008902	0:48	1108
	696932	1044739	27:02	2387220	1:31	14029
	736400	5331790	126:46	12316618	5:17	48577
	777488	870204	21:36	1887252	2:02	13872
	1859160	1879786	49:04	4226714	6:00	25396
Generated, with cycle						
	18699	22449	0:06	22	0:05	68
	33400	2073288	0:37	30824	0:24	555
	46956	83110	0:05	108	0:09	702
	448875	1863905	0:51	21106	0:56	3435
Random, without cycle						
	4000	353247	14:03	1390262	0:17	17868
	5000	839679	31:48	3151724	0:32	2489
	80000	522327	30:11	2042212	1:39	87002
	60000	1111411	57:19	4131210	4:08	98686
	947200	5781959	184:23	13338462	9:49	47030
Random, with cycle						
	18000	1169438	1:20	104822	0:09	862
Philosophers						
(12)	94578	42154	2:06	168616	0:13	756
(14)	608185	269923	16:11	1079692	1:40	4500

Experiments show that in spite of worse theoretical worst time complexity of *NC* algorithm its behaviour in practice can outperform the theoretically better *NDFS* one. This is due to the number of communications which has essential impact on the resulting time. In *NC* algorithm the messages can be grouped into packets and sent together. It is a general experience that the time needed for delivering t single messages is much higher than the time needed for delivering those messages grouped into one packet. On the other hand, *NDFS* algorithm does not admit such a grouping. Another disadvantage of *NDFS* is that during the passing of messages all the processors are idle, while in *NC* algorithm the computation can continue immediately after sending a message. Last but not least, in *NDFS* all but one processor are idle whereas in *NC* all can compute concurrently. We notice that all mentioned advantages of *NC* algorithm demonstrate themselves especially for systems without cycles where the whole graph has to be searched. This is in fact the desired property of our algorithm as the state explosion demonstrates itself just in these cases. Both algorithms perform equally well on graphs with cycles.

We have accomplished yet another set of tests in order to validate the scalability of the *NC* algorithm. The tests confirm that it scales well, i.e. the overall time needed for treating a graph is decreasing as the number of involved processors is increased.

6 Conclusions

Parallel and distributed algorithms for reachability analysis and model checking have recently been investigated as a possible method to handle large state spaces. The core problem of automata based model checking is the detection of reachable accepting cycles in the state space. The classical depth first strategy provides a suitable approach to cycle detection in a sequential case. However, the depth first search approach is difficult to distribute.

The paper proposes a novel approach to the cycle detection problem in a distributed environment. The main idea is to transform the accepting cycle detection problem to the single-source shortest path problem in graphs with real number edge lengths – negative cycle problem. We have proposed a scalable distributed algorithm to solve this problem and we have performed a series of experiments to evaluate its performance.

The performance of the algorithm was compared with a distributed DFS based algorithm. The experimental results show that the distributed algorithm based on negative cycle detection significantly outperforms the DFS based one due to higher degree of asynchronous parallelism which allows to optimise necessary communication. DFS based algorithms rely on strict synchronisation.

In the future we aim to embed the algorithm in a suitable automata based verification tool (e.g. SPIN) to be able to test its applicability to a non-trivial series of real systems. Furthermore, we intend to explore various heuristics and implementation techniques to optimise its performance.

References

1. S. Aggarwal, R. Alonso, and C. Courcoubetis. Distributed reachability analysis for protocol verification environments. In *Discrete Event Systems: Models and Application*, volume 103 of *LNCS*, pages 40–56. Springer, 1987.
2. J. Barnat, L. Brim, and J. Stříbrná. Distributed LTL Model-Checking in SPIN. In *Proc. SPIN 2001*, volume 2057 of *LNCS*, pages 200–216. Springer, 2001.
3. G. Behrmann, T. S. Hune, and F. W. Vaandrager. Distributed timed model checking — how the search order matters. In *Proc. CAV 2000*, volume 1855 of *LNCS*, pages 216–231. Springer, 2000.
4. S. Ben-David, T. Heyman, O. Grumberg, and A. Schuster. Scalable distributed on-the-fly symbolic model checking. In *Proc. FMCAD 2000*, 2000.
5. B. Bollig, M. Leucker, and M Weber. Parallel model checking for the alternation free mu-calculus. In *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 543–558. Springer, 2001.
6. L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed shortest path for directed graphs with negative edge lengths. Technical Report FIMU-RS-2001-01, Faculty of Informatics, Masaryk University Brno, http://www.fi.muni.cz/informatics/reports/, 2001.
7. S. Chaudhuri and C. D. Zaroliagis. Shortest path queries in digraphs of small treewidth. In *Proc. ESA 1995*, volume 979 of *LNCS*, pages 31–45. Springer, 1995.
8. B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, (85):277–311, 1999.

9. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Progress on the state explosion problem in model checking. In *Informatics - 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 176–194. Springer, 2001.
10. E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. *Journal of Algorithms*, 21(2):331–357, 1996.
11. T. H. Cormen, Ch. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT, 1990.
12. A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra's shortest path algorithm. In *Proc. MFCS 1998*, volume 1450 of *LNCS*, pages 722–731. Springer, 1998.
13. P. Spirakis D. Kavvadias, G. Pantziou and C. Zaroliagis. Efficient sequential and parallel algorithms for the negative cycle problem. In *Proc. ISAAC 1994*, volume 834 of *LNCS*, pages 270–278. Springer, 1994.
14. O. Grumberg, T. Heyman, and A. Schuster. Distributed model checking for mu-calculus. In *Proc. 13th Conference on Computer-Aided Verification CAV01*, LNCS. Springer, 2001.
15. T. Heyman, D. Geist, O. Grumberg, and A. Schuster. Achieving scalability in parallel reachability analysis of very large circuits. In *Proc. CAV 2000*, volume 1855 of *LNCS*, pages 20–35. Springer, 2000.
16. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
17. G.J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *The Spin Verification System*, pages 23–32. American Mathematical Society, 1996.
18. F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. SPIN 1999*, number 1680 in LNCS. Springer, 1999.
19. U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *Proc. EUROPAR 2000*. LNCS, 2000.
20. K. Ramarao and S. Venkatesan. On finding and updating shortest paths distributively. *Journal of Algorithms*, 13:235–257, 1992.
21. J.H. Reif. Depth-first search is inherrently sequential. *Information Processing Letters*, 20(5):229–234, 1985.
22. S.H. Roosta. *Parallel processing and parallel algorithms*. Springer, 2000.
23. U. Stern and D.L. Dill. Parallelizing the Murφ verifier. In *Proc. CAV 1997*, volume 1254 of *LNCS*, pages 256–267. Springer, 1997.
24. R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on computing*, pages 146–160, 1972.
25. J. Traff and C.D. Zaroliagis. A simple parallel algorithm for the single-source shortest path problem on planar digraphs. In *Proc. IRREGULAR-3 1996*, volume 1117 of *LNCS*, pages 183–194S. Springer, 1996.
26. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS 1986*, pages 332–344. Computer Society Press, 1986.

Computability and Complexity Results for a Spatial Assertion Language for Data Structures

Cristiano Calcagno[1,2], Hongseok Yang[3], and Peter W. O'Hearn[1]

[1] Queen Mary, University of London
[2] DISI, University of Genova
[3] ROPAS, KAIST

Abstract. This paper studies a recently developed an approach to reasoning about mutable data structures, which uses an assertion language with spatial conjunction and implication connectives. We investigate computability and complexity properties of a subset of the language, which allows statements about the shape of pointer structures (such as "there is a link from x to y") to be made, but not statements about the data held in cells (such as "x is a prime number"). We show that validity, even for this restricted language, is not r.e., but that the quantifier-free sublanguage is decidable. We then consider the complexity of model checking and validity for several fragments.

1 Introduction

This paper studies a recently developed an approach to reasoning about mutable data structures [9,5]. The assertion language includes spatial conjunction and implication connectives alongside those of classical logic, in the style of the logic of Bunched Implications [8]. The conjunction $P * Q$ is true just when the current heap can be split into disjoint components, one of which makes P true and the other of which makes Q true. The implication $P -\!\!* Q$ says that whenever P is true for a new or fresh piece of heap, Q is true for the combined new and old heap. In addition, there is an atomic formula, the points-to relation $E \mapsto F, G$, which says that E points to a cons cell holding F in its car and G in its cdr.

As a small example of $*$,

$$(x \mapsto a, y) * (y \mapsto b, x)$$

describes a two-element circular linked list, with a and b in the data fields. The conjunction $*$ here requires x and y to be pointers to distinct and non-overlapping cells. For an example of $-\!\!*$,

$$(x \mapsto a, b) * ((x \mapsto c, b) -\!\!* P)$$

says that x points to a cell holding (a, b), and that P will hold if we update the car to c.

The logic of [9,5,7] can be used to structure arguments in a way that leads to pleasantly simple proofs of pointer algorithms. But the assertion language

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 108–119, 2001.
© Springer-Verlag Berlin Heidelberg 2001

that the logic uses to describe pre and postconditions is itself new, and its properties have not been studied in detail. The purpose of this paper is to study computability and complexity problems for the language.

We consider a pared down sublanguage, which includes the points-to relation and equality as atomic predicates, but not arithmetic or other expressions or atomic predicates for describing data. We do this to separate out questions about the shapes of data structures themselves from properties of the data held in them. This also insulates us from decidability questions about the data. In our language we can write a formula that says that x points to a linked list with two nodes, but not a formula that says that the list is sorted.

Our first result is that, even with these restrictions, the question of validity is not r.e. The spatial connectives are not needed for this negative result. This result might seem somewhat surprising, given the sparseness of the language; decidability would obtain immediately were we to omit the points-to relation. The proof goes by reduction from a well-known non-r.e. problem of finite model theory: deciding whether a closed first-order logic formula holds for all nonempty *finite* structures.

This result has two consequences. The first is that it tells us that we cannot hope to find an axiomatic description of $\mapsto$, adequate to the whole language. The second is that we should look to sublanguages if we are to find a decidability result.

Our second result is that the quantifier-free sublanguage is decidable. The main subtlety in the proof is the treatment of $-\!\!*$, whose semantics uses a universal quantification over heaps. This is dealt with by a bounding result, which restricts the number of heaps that have to be considered to verify or falsify a formula.

We then consider the complexity of model checking and validity. For the quantifier-free fragment and several sublanguages both questions are shown to be PSPACE-complete. One fragment is described where the former is NP-complete and the latter Π_2^{P}-complete. We also remark on cases where (like in propositional calculus) model checking is linear and validity coNP-complete.

2 The Model and the Assertion Language

In this section we present a spatial assertion language and its semantics. The other sections study properties of fragments of this language.

Throughout the paper we will use the following notation. A finite map f from X to Y is written $f\colon X \rightharpoonup_{fin} Y$, and $dom(f)$ indicates the domain of f. The notation $f\#g$ means that f and g have disjoint domains, and in that case $f * g$ is defined by $(f * g)(x) = y$ iff $f(x) = y$ or $g(x) = y$.

The syntax of expressions E and assertions P for binary heap cells is given by the following grammar:

$$E ::= x, y \ldots \mid \mathsf{nil}$$
$$P ::= (E \mapsto E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \forall x.\, P \mid \mathsf{emp} \mid P * P \mid P -\!\!* P$$

Expressions are either variables or the constant nil. Assertions include equality, usual connectives from first-order classical logic, and spatial connectives. The predicate $(E \mapsto E_1, E_2)$ asserts that E is the only allocated cell and it points to a binary heap cell containing E_i in the i-th component. The assertion emp says that the heap is empty. The assertion $P_1 * P_2$ means that it is possible to split the current heap in disjoint sub-heaps making the two assertions true. The assertion $P_1 -\!\!* P_2$ means that for each new heap disjoint from the current one and making P_1 true, the combined new and old heap makes P_2 true.

The other logical connectives are expressible as usual as derived notation:

$$\neg P \stackrel{\Delta}{=} P \Rightarrow \mathsf{false} \qquad P_1 \wedge P_2 \stackrel{\Delta}{=} \neg(P_1 \Rightarrow \neg P_2) \qquad \exists x.\, P \stackrel{\Delta}{=} \neg(\forall x.\, \neg P)$$

Expressions and assertions for binary heap cells are interpreted in the following model:

$$\begin{aligned} \mathit{Val} &\stackrel{\Delta}{=} \mathit{Loc} \cup \{\mathit{nil}\} \\ \mathit{Stack} &\stackrel{\Delta}{=} \mathit{Var} \rightarrow \mathit{Val} \\ \mathit{Heap} &\stackrel{\Delta}{=} \mathit{Loc} \rightharpoonup_{\mathit{fin}} \mathit{Val} \times \mathit{Val} \\ \mathit{State} &\stackrel{\Delta}{=} \mathit{Stack} \times \mathit{Heap} \end{aligned}$$

Values are either locations or *nil*, and a state is composed of a stack and a heap. The heap is a finite map from locations to binary heap cells, whose domain indicates the locations that are allocated at the moment. The semantics of expressions and assertions is given in Table 1.

Definition 1 (Validity). *We say that P is valid, written $\models P$, if $s, h \models P$ for all the states (s, h).*

Table 1. Semantics of Expressions and Assertions

$$\begin{aligned} [\![x]\!]s &\stackrel{\Delta}{=} s(x) \\ [\![\mathsf{nil}]\!]s &\stackrel{\Delta}{=} \mathit{nil} \end{aligned}$$

$s, h \models (E \mapsto E_1, E_2)$	iff $\mathit{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = ([\![E_1]\!]s, [\![E_2]\!]s)$
$s, h \models E_1 = E_2$	iff $[\![E_1]\!]s = [\![E_2]\!]s$
$s, h \models \mathsf{false}$	never
$s, h \models P_1 \Rightarrow P_2$	iff if $s, h \models P_1$ then $s, h \models P_2$
$s, h \models \mathsf{emp}$	iff $\mathit{dom}(h) = \emptyset$
$s, h \models P_1 * P_2$	iff there exist h_1 and h_2 such that $h_1 \# h_2;\ h_1 * h_2 = h;\ s, h_1 \models P_1;\ s, h_2 \models P_2$
$s, h \models P_1 -\!\!* P_2$	iff for all h_1 such that $h \# h_1$ and $(s, h_1) \models P_1$, $(s, h * h_1) \models P_2$
$s, h \models \forall x.\, P$	iff for any v in $\mathit{Val}, s[x \mapsto v], h \models P$

3 Undecidability

The main result in this section is that the validity problem is not recursively enumerable even when the spatial connectives, $*$, emp and $-\!\!*$, do not appear in assertions.

Theorem 1. *Deciding whether an assertion is valid is not recursively enumerable even when the assertion language is restricted as follows:*

$$P ::= (E \hookrightarrow E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \forall x.\, P$$

where $(E \hookrightarrow E_1, E_2)$ *is* $(E \mapsto E_1, E_2) * \mathsf{true}$.

Note that the theorem uses an intuitionistic variant $\hookrightarrow$ of the predicate $\mapsto$ because only with the $\mapsto$ predicate, we can not express that a heap cell l is allocated and contains (v_1, v_2) without requiring that l is the only allocated heap cell. The meaning of $(E \hookrightarrow E_1, E_2)$ is that a heap cell E is allocated and contains (E_1, E_2) but it need not be the only allocated cell.

We prove the theorem by reducing the validity on nonempty finite structures of closed first-order logic formulas to validity for our restricted language. Then, the conclusion follows by a standard result from finite model theory [3]:

Theorem 2 (Trakhtenbrot). *Even if a signature consists only of one binary relation, the set of closed first-order logic formulas valid on all nonempty finite structures is not recursively enumerable.*

The reduction goes by translating a first-order logic formula with a single binary relation R to an assertion. Let φ be a first-order logic formula, which is not necessarily closed. The translation $rd(\varphi)$ is given as follows:

$$rd(\varphi) \quad \triangleq (\exists x.\, (x \hookrightarrow \mathsf{nil}, \mathsf{nil})) \Rightarrow prd(\varphi)$$

$$\begin{aligned}
prd(R(x,y)) &\triangleq (\exists z.\, (z \hookrightarrow x, y)) \wedge (x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \wedge (y \hookrightarrow \mathsf{nil}, \mathsf{nil}) \\
prd(\varphi \Rightarrow \psi) &\triangleq prd(\varphi) \Rightarrow prd(\psi) \\
prd(\mathsf{false}) &\triangleq \mathsf{false} \\
prd(x = y) &\triangleq (x = y) \wedge (x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \\
prd(\exists x.\, \varphi) &\triangleq \exists x.\, ((x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \wedge prd(\varphi))
\end{aligned}$$

Intuitively, the translation encodes the relation $R^{\mathcal{A}}$ and the universe $|\mathcal{A}|$ of a nonempty finite structure $\mathcal{A}$ by heap cells: each element in $|\mathcal{A}|$ is encoded as an allocated cell containing (nil, nil), and a related pair (a_1, a_2) in $R^{\mathcal{A}}$ is encoded as an allocated cell containing (x, y) where x and y are encodings of a_1 and a_2, respectively. Note that the guard $(\exists x.\, (x \hookrightarrow \mathsf{nil}, \mathsf{nil}))$ in the definition of $rd(\varphi)$ models the fact that the universe of a finite structure must be nonempty.

The reduction becomes complete once we prove that for *closed* first-order logic formulas, the translation preserves and reflects validity. To show that validity is reflected, we prove a lemma which implies that for all nonempty finite

structures $\mathcal{A}$ and environments η (mapping variables to elements of $|\mathcal{A}|$), it is always possible to find a state (s, h) so that

$$\mathcal{A}, \eta \models \varphi \iff (s, h) \models rd(\varphi) \quad \text{for all closed first-order formulas } \varphi.$$

Lemma 1. *Let $\mathcal{A}$ be a nonempty finite structure for the signature $\{R\}$, where R is a binary relation. For all heaps h and sets B, C of locations such that*

- *$\{B, C\}$ is a partition of $dom(h)$;*
- *γ is a bijection from $|\mathcal{A}|$ to B such that $h(\gamma(a)) = (nil, nil)$ for all $a \in |\mathcal{A}|$; and*
- *δ is a bijection from $R^{\mathcal{A}}$ to C such that $h(\delta(a_1, a_2)) = (\gamma(a_1), \gamma(a_2))$ for all $(a_1, a_2) \in R^{\mathcal{A}}$,*

we have

$$\mathcal{A}, \eta \models \varphi \iff \gamma \circ \eta, h \models rd(\varphi)$$

for all first-order formulas φ and environments η.

Proof. Since the universe $|\mathcal{A}|$ is not empty, the guard $(\exists x.\, (x \hookrightarrow \mathsf{nil}, \mathsf{nil}))$ holds for the state (s, h). So, it suffices to prove the following claim: for all first-order formulas φ,

$$\mathcal{A}, \eta \models \varphi \iff \gamma \circ \eta, h \models prd(\varphi)$$

It is straightforward to show the claim using induction over the structure of φ. □

Before showing that validity is preserved by the translation, we note that when φ is closed, so is $rd(\varphi)$; so, $rd(\varphi)$ is valid if and only if for all heaps h, there is some stack s with $(s, h) \models rd(\varphi)$. Let φ be a closed first-order formula and let h be a heap. When h does not have any cells containing (nil, nil), the guard $(\exists x.\, (x \hookrightarrow nil, nil))$ of $rd(\varphi)$ always becomes false; consequently, $(s, h) \models rd(\varphi)$ for all stacks s. The key idea to handle the other case, where a heap h has at least one cell containing (nil, nil), is to build a nonempty finite structure $\mathcal{A}$ and a stack s such that φ holds in $\mathcal{A}$ if and only if $(s, h) \models rd(\varphi)$. We construct such a stack simply by mapping all variables to the address of allocated cells in h containing (nil, nil); then, the following lemma shows how to construct the needed structure.

Lemma 2. *For all heaps h and stacks s such that $h(s(x))$ is defined and equal to (nil, nil) for all variables x, let $\mathcal{A}$ be a structure for the signature $\{R\}$ given by:*

- *$|\mathcal{A}| = \{l \in dom(h) \mid h(l) = (nil, nil)\}$; and*
- *$(l_1, l_2) \in R^{\mathcal{A}}$ iff l_1, l_2 are in $|\mathcal{A}|$ and $h(l) = (l_1, l_2)$ for some $l \in dom(h)$.*

Then, $\mathcal{A}, s \models \varphi$ iff $s, h \models rd(\varphi)$.

Proof. Note that the structure $|\mathcal{A}|$ can not be empty because in h, at least a single allocated heap cell must contain (nil, nil); otherwise, no stack would satisfy the condition in the lemma. One consequence of this fact is that $(s,h) \models rd(\varphi)$ iff $(s,h) \models prd(\varphi)$. So, it suffices to show that

$$\mathcal{A}, s \models \varphi \Longleftrightarrow s, h \models prd(\varphi),$$

which can be easily proved using induction over φ. □

4 Decidable Fragment

The undecidability result in the previous section indicates that in order to obtain a decidable fragment of the assertion language, either quantifiers must be taken out in the fragment or they should be used in a restricted manner. In this section, we consider the quantifier-free fragment of the assertion language, including spatial connectives, emp, $*$ and $-\!\!*$. The main result in the section is:

Theorem 3. *Deciding the validity of assertions is algorithmically decidable as long as the assertions are instances of the following grammar:*

$$P ::= (E \mapsto E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \mathsf{emp} \mid P * P \mid P -\!\!* P$$

To prove the theorem, we need to show that there is an algorithm which takes an assertion following the grammar in the theorem and answers whether the assertion holds for all states. The main observation is that each assertion determines a finite set of states so that if the assertion holds for all states in the set, it indeed holds for all the states. The proof proceeds in two steps: first we consider the case that an assertion P and a state s, h are given so that an algorithm is supposed to answer whether $s, h \models P$; then, we construct an algorithm which, given an assertion P, answers whether $s, h \models P$ holds for all the states (s, h). In the remainder of the section, we assume that all the assertions follow the grammar given in Theorem 3.

The problem of algorithmically deciding whether $s, h \models P$ holds given P, s, h as inputs is not as straightforward as it seems because of $-\!\!*$: when P is of the form $Q -\!\!* R$, the interpretation of $s, h \models P$ involves quantification over all heaps, which might require to check infinite possibilities. So, the decidability proof is mainly for showing that there is a finite boundary algorithmically determined by Q and R. We first define the *size* of an assertion, which is used to give an algorithm to determine the boundary.

Definition 2 (size of P). *For an assertion P, we define* size *of P, $|P|$, as follows:*

$$\begin{array}{rcl@{\qquad}rcl} |(E \mapsto E_1, E_2)| & = & 1 & |E_1 = E_2| & = & 0 \\ |\mathsf{false}| & = & 0 & |P \Rightarrow Q| & = & max(|P|, |Q|) \\ |P * Q| & = & |P| + |Q| & |P -\!\!* Q| & = & |Q| \\ |\mathsf{emp}| & = & 1 & & & \end{array}$$

The size of P determines a bound on the number of heap cells we have to consider to determine whether P is true or not. For instance, the size of $(x \mapsto \mathsf{nil}, \mathsf{nil}) * (y \mapsto \mathsf{nil}, \mathsf{nil})$ is 2; and to decide whether it or its negation is true, or whether it is merely satisfiable, requires us only to look at heaps of size upto 2.

The following proposition claims that there is a bound number of heaps to check in the interpretation of $s, h \models Q \mathrel{-\!\!*} R$; the decidability result is just an immediate corollary. Let ord be an effective enumeration of Loc.

Proposition 1. *Given a state (s, h) and assertions Q, R, let X be $FV(Q) \cup FV(R)$ and B a finite set consisting of the first $max(|Q|, |R|)$ locations in $Loc - (dom(h) \cup s(X))$ where the ordering is given by ord. Pick a value $v \in Val - s(X) - \{nil\}$. Then, $(s, h) \models Q \mathrel{-\!\!*} R$ holds iff for all h_1 such that*

- *$h \# h_1$ and $(s, h_1) \models Q$;*
- *$dom(h_1) \subseteq B \cup s(X)$; and*
- *for all $l \in dom(h_1)$, $h_1(l) \in (s(X) \cup \{nil, v\}) \times (s(X) \cup \{nil, v\})$*

we have that $(s, h * h_1) \models R$.

To see why the proposition implies the decidability result, notice that there are only finitely many h_1's satisfying the conditions because both $B \cup s(X)$ and $s(X) \cup \{nil, v\}$ are finite. Since all the other cases of P only involve finitely many ways to satisfy $s, h \models P$, the exhaustive search gives the decision algorithm.

The interesting direction of the proposition is "if" because the only-if direction follows from the interpretation of $\mathrel{-\!\!*}$. Intuitively, the if direction of the proposition holds because the following three changes of heap cells do not affect the truth of either Q or R: relocating "garbage" heap cells (those not in $s(X)$); de-allocating redundant garbage heap cells when there are more than $max(|Q|, |R|)$ of them; overwriting "uninteresting values" (those not in $s(X) \cup \{nil\}$) by another uninteresting value (v). Then, for every heap h_1' with $h \# h_1'$, there is a sequence of such changes which transforms h_1' and $h * h_1'$ to h_1 and $h * h_1$, respectively, such that h_1 satisfies the last two conditions in the proposition. The proposition follows because each step in the sequence preserves the truth of both Q and R; so, $(s, h_1') \models Q$ implies $(s, h_1) \models Q$, and $(s, h_1 * h) \models R$ implies $(s, h_1' * h) \models R$.

Corollary 1. *Given a stack s and an assertion P, checking $(s, h) \models P$ for all h is decidable.*

Proof. The corollary holds because $s, h \models P$ for all h iff $s, [] \models (\neg P) \mathrel{-\!\!*} \mathsf{false}$. □

For the decidability of checking $(s, h) \models P$ for all states (s, h), we observe that the actual values of variables are not relevant to the truth of an assertion as long as the "relationship" of the values remains the same. We define a relation $\approx_X$ to capture this "relationship" formally. Intuitively, two states are related by $\approx_X$ iff the relationship of the values, which are stored in variables in X or in heap cells, are the same in the two states.

Definition 3 ($\approx_X$). *For states (s,h) and (s',h') and a subset X of Var, $(s,h) \approx_X (s',h')$ iff there exists a bijection r from Val to Val such that $r(nil) = nil$; $r(s(x)) = s'(x)$ for all $x \in X$; and $(r \times r)(h(l)) = h'(r(l))$ for all $l \in Loc$.* [1]

Proposition 2. *For all the states (s,h) and (s',h') and all assertions P such that $(s,h) \approx_{FV(P)} (s',h')$, if $(s,h) \models P$, then $(s',h') \models P$.*

Lemma 3. *Given a state (s,h) and an assertion P, let B be the set consisting of the first $|FV(P)|$ locations in Loc, where the ordering is given by* ord*. Then, there exists a state (s',h') such that $s'(Var - FV(P)) \subseteq \{nil\}$; $s'(FV(P)) \subseteq B \cup \{nil\}$; and $(s,h) \approx_{FV(P)} (s',h')$.*

The decidability result follows from the above lemma. To see the reason, we note that because of the lemma, for all assertions P, there is a finite set of stacks such that if for all stacks s in the set and all heaps h, $(s,h) \models P$, then P holds for all states whose stack is not necessarily in the set. Therefore, a decision algorithm is obtained by exhaustively checking for each stack s in the finite set whether $(s,h) \models P$ holds for all heaps h using the algorithm in Corollary 1.

Corollary 2. *Given an assertion P, checking $(s,h) \models P$ for all the states (s,h) is decidable.*

5 Complexity

In this section we study the complexity of model checking for some fragments of the decidable logic of Section 4.

We consider the following fragments, where $(E \not\mapsto -)$ means that E is not allocated ($s,h \models (E \not\mapsto -)$ iff $[\![E]\!]s \notin dom(h)$):

Language		MC	VAL
$\mathcal{L}$	$P ::= (E \mapsto E, E) \mid (E \not\mapsto -) \mid E = E \mid E \neq E \mid \mathsf{false}$ $\mid P \wedge P \mid P \vee P \mid \mathsf{emp}$	P	coNP
$\mathcal{L}^*$	$P ::= \mathcal{L} \mid P * P$	NP	Π^{P}_2
$\mathcal{L}^{\neg *}$	$P ::= \mathcal{L} \mid \neg P \mid P * P$	PSPACE	PSPACE
$\mathcal{L}^{-\!\!*}$	$P ::= \mathcal{L} \mid P -\!\!* P$	PSPACE	PSPACE
$\mathcal{L}^{\neg * -\!\!*}$	$P ::= \mathcal{L} \mid \neg P \mid P * P \mid P -\!\!* P$	PSPACE	PSPACE

Given a fragment $\mathcal{L}^c$, the corresponding model-checking problem $MC(\mathcal{L}^c)$ is deciding whether $s,h \models P$ holds given a state (s,h) and an assertion $P \in \mathcal{L}^c$. The validity problem asks whether a formula is true in all states. In the above table the second-last column reports the complexity of model checking and the last the complexity of validity.

[1] the equality in $(r \times r)(h(l)) = h'(r(l))$ means that if one side of the equation is defined, the other side is also defined and they are equal.

The easy fragment is $\mathcal{L}$. Clearly $MC(\mathcal{L})$ can be solved in linear time by the obvious algorithm arising from the semantic definitions, and it is not difficult to show that the validity is coNP-complete. As soon as we add $*$, model checking bumps up to NP-complete. The validity problem for $\mathcal{L}^*$ is Π_2^{P}-complete; we show the former but consideration of the latter is omitted for brevity. It is possible to retain linear model checking when $*$ is restricted so that one conjunct is of the form $(E \mapsto E_1, E_2)$. The fragment $\mathcal{L}^{\neg * \neg *}$ is the object of the decidability result of Section 4; a consequence of our results there is that model checking and validity can be decided in polynomial space. Below we show PSPACE-hardness for model checking for the two fragments $\mathcal{L}^{\neg *}$ and $\mathcal{L}^{-*}$. It is a short step to show PSPACE-hardness for validity.

5.1 $MC(\mathcal{L}^*)$ Is NP-Complete

In this section we show directly that $MC(\mathcal{L}^*)$ belongs to NP, and give a reduction from an NP-complete problem to it.

Proposition 3. *$MC(\mathcal{L}^*)$ is in NP.*

Proof. The only interesting part is deciding whether $s, h \models P * Q$ holds. The algorithm proceeds by choosing non-deterministically a set $D \subseteq dom(h)$, determining a splitting of h in two heaps h_1 and h_2 obtained by restricting h to D and to $dom(h) - D$ respectively. □

Definition 4. *The problem SAT is, given a formula F from the grammar*

$$F ::= x \mid \neg x \mid F \wedge F \mid F \vee F$$

deciding whether it is satisfiable, i.e. whether there exists an assignment of boolean values to the free variables of F making F true.

Definition 5. *The translation from formulas F to assertions P of $\mathcal{L}^*$ is defined by a function $tr(-)$.*[2]

$$tr(x) \stackrel{\Delta}{=} (x \mapsto \mathsf{nil}, \mathsf{nil}) * \mathsf{true} \qquad tr(\neg x) \stackrel{\Delta}{=} (x \not\mapsto -)$$
$$tr(F_1 \wedge F_2) \stackrel{\Delta}{=} tr(F_1) \wedge tr(F_2) \qquad tr(F_1 \vee F_2) \stackrel{\Delta}{=} tr(F_1) \vee tr(F_2)$$

Proposition 4. *A formula F with variables $\{x_1, \ldots, x_n\}$ is satisfiable if and only if $s_0, h_0 \models tr(F) * \mathsf{true}$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, $dom(h_0) = \{l_1, \ldots, l_n\}$ and $h_0(l_i) = (nil, nil)$ for $i = 1, \ldots, n$.*

Proof. The truth of a boolean variable x is represented by its being allocated; in the initial state (s_0, h_0) all the variables are allocated. The formula $tr(F) * \mathsf{true}$ is true if and only if there exists a subheap h' making $tr(F)$ true, and subheaps correspond to assignments of boolean values to the variables in F. □

Since the translation and construction of (s_0, h_0) can be performed in polynomial time, an immediate consequence is NP-hardness of $MC(\mathcal{L}^*)$, hence NP-completeness.

[2] true can be expressed by $\mathsf{nil} = \mathsf{nil}$ in $\mathcal{L}^*$.

5.2 $MC(\mathcal{L}^{\neg *})$ Is PSPACE-Complete

In this section PSPACE-hardness of $MC(\mathcal{L}^{\neg *})$ is proved by reducing a PSPACE-complete problem to it. Completeness follows from the fact that $MC(\mathcal{L}^{\neg *-\!\!*})$ is in PSPACE and that $\mathcal{L}^{\neg *}$ is a sub-fragment of $\mathcal{L}^{\neg *-\!\!*}$.

Definition 6. *The problem QSAT is, given a closed formula G from the grammar*

$$F ::= x \mid \neg x \mid F \wedge F \mid F \vee F, \qquad G ::= \forall x_1.\exists y_1. \ldots .\forall x_n.\exists y_n.F$$

deciding whether it is true.

Definition 7. *The translation from formulas G to assertions P of $\mathcal{L}^{\neg *}$ is defined by a function $tr(-)$:*

$$\begin{aligned} tr(x) &\stackrel{\Delta}{=} (x \mapsto \mathsf{nil}, \mathsf{nil}) * \mathsf{true} & tr(\neg x) &\stackrel{\Delta}{=} (x \not\mapsto -) \\ tr(F_1 \wedge F_2) &\stackrel{\Delta}{=} tr(F_1) \wedge tr(F_2) & tr(F_1 \vee F_2) &\stackrel{\Delta}{=} tr(F_1) \vee tr(F_2) \\ tr(\exists y_i.G) &\stackrel{\Delta}{=} ((y_i \mapsto \mathsf{nil}, \mathsf{nil}) \vee \mathsf{emp}) * tr(G) \\ tr(\forall x_i.G) &\stackrel{\Delta}{=} \neg\,(((x_i \mapsto \mathsf{nil}, \mathsf{nil}) \vee \mathsf{emp}) * \neg\, tr(G)) \end{aligned}$$

Proposition 5. *A closed formula G is true if and only if $s_0, h_0 \models tr(G)$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, $dom(h_0) = \{l_1, \ldots, l_n\}$ and $h_0(l_i) = (nil, nil)$ for $i = 1, \ldots, n$.*

Proof. The truth of a boolean variable x is represented by its being allocated; in the initial state (s_0, h_0) all the variables are allocated. The only interesting cases are the quantifiers. The invariant is that $tr(\exists y_i.G)$ is checked in a state where y_i is allocated, thus $((y_i \mapsto \mathsf{nil}, \mathsf{nil}) \vee \mathsf{emp}) * tr(G)$ holds iff $tr(G)$ holds either for the current state or for the state obtained by de-allocating y_i. In other words, G either holds for y_i true or for y_i false. The translation of $(\forall x_i.\, -)$ is essentially $\neg(\exists x_i.\neg\, -)$. □

Observing that the translation and construction of (s_0, h_0) can be performed in polynomial time, we have shown PSPACE-hardness of $MC(\mathcal{L}^{\neg *})$.

5.3 $MC(\mathcal{L}^{-\!\!*})$ Is PSPACE-Complete

In analogy with the previous section, a translation from QSAT to $MC(\mathcal{L}^{-\!\!*})$ is presented.

This case is more complicated, since $-\!\!*$ provides a natural way of representing universal quantifiers, but there is no immediate way to represent existentials. Our solution is to use two variables x_t and x_f to represent a boolean variable x. There are three admissible states:

- initial, when neither x_t nor x_f is allocated;

– true, when x_t is allocated and x_f is not;
– false, when x_f is allocated and x_t is not.

We use some auxiliary predicates:

$$\begin{aligned}(x \hookrightarrow -) &\triangleq ((x \mapsto \mathsf{nil}, \mathsf{nil}) \mathrel{-\!\!*} \mathsf{false}) \wedge (x \neq \mathsf{nil}) \\ I_x &\triangleq (x_t \not\hookrightarrow -) \wedge (x_f \not\hookrightarrow -) \\ OK_x &\triangleq ((x_t \hookrightarrow -) \wedge (x_f \not\hookrightarrow -)) \vee ((x_f \hookrightarrow -) \wedge (x_t \not\hookrightarrow -))\end{aligned}$$

The meaning of $(x \hookrightarrow -)$ is that it is not possible to extend the current heap with x pointing to $(\mathsf{nil}, \mathsf{nil})$, i.e. x is allocated; I_x means that x is in an initial state, and OK_x means that x is either in state true or in state false.

Definition 8. *Given a closed formula $\forall x_1.\exists y_1. \ldots .\forall x_n.\exists y_n.F$, define the ordered set $V \triangleq \{x_1 < y_1 < \ldots < x_n < y_n\}$. Write S^{op} for $V - S$ when $S \subseteq V$. Define $\{\leq x\} \triangleq \{x' \in V | x' \leq x\}$. The predicates are extended as follows:*

$$I_S \triangleq \bigwedge_{x \in S} I_x \qquad\qquad OK_S \triangleq \bigwedge_{x \in S} OK_x$$

The translation is defined by a function $tr(-)$:

$$\begin{aligned}tr(x) &\triangleq (x_t \hookrightarrow -) \\ tr(\neg x) &\triangleq (x_f \hookrightarrow -) \\ tr(F_1 \wedge F_2) &\triangleq tr(F_1) \wedge tr(F_2) \\ tr(F_1 \vee F_2) &\triangleq tr(F_1) \vee tr(F_2) \\ tr(\forall x_i.\, G) &\triangleq (OK_{\{x_i\}} \wedge I_{\{x_i\}^{op}}) \mathrel{-\!\!*} tr(G) \\ tr(\exists y_i.G) &\triangleq \sim((OK_{\{\leq x_i\}} \wedge I_{\{\geq y_i\}}) \wedge \sim(OK_{\{\leq y_i\}} \wedge I_{\{\geq x_{i+1}\}} \wedge tr(G)))\end{aligned}$$

where $\sim P$ is short for $P \mathrel{-\!\!*} \mathsf{false}$.

Intuitively, the translation of x says that x is in state true, and the translation of $\neg x$ says that x is in state false. For $(\forall x_i.\, G)$, the invariant is that $OK_{\{\leq y_{i-1}\}}$ and $I_{\geq x_i}$ hold, and the translation says that G holds after extending the current heap with any new heap containing only x_i in an OK state (i.e. true or false). For $(\exists y_i.\, G)$, the invariant is that $OK_{\{\leq x_i\}}$ and $I_{\geq y_i}$ hold. The formula $\sim(P \wedge Q)$ implies that when P holds in a new heap, the heap does not satisfy Q; in particular, if P is $OK_{\{\leq x_i\}} \wedge I_{\{\geq y_i\}}$ and P holds in the current heap, $\sim(P \wedge Q)$ implies that inverting the boolean value of x makes Q false. This case is the most complicated of the translation, and involves a double negation. In words, it says that given an initial heap h_0, inverting the boolean values of variables in $\{\leq x_i\}$ leads to a heap h_1 which makes the following false: for every heap h_2 obtained from h_1 by inverting again the boolean values of variables in $\{\leq x_i\}$ and by assigning some boolean value to y_i, G does not hold in h_2.

Proposition 6. *A closed formula G is true if and only if $s_0, [] \models tr(G)$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, and $[]$ is the empty heap.*

To obtain the PSPACE-hardness result, observe that the translation can be performed in polynomial time, since the size of each I_S and OK_S is linear in the number of variables.

6 Future Work

Possible directions for future work include incorporating heap variables that allow us to take snapshots of the heap, with suitable restrictions [6] to maintain decidability, and also recursive definitions or special atomic predicates [1] for describing paths through the heap. We also plan to investigate the relation of our approach to work on model checking mobile ambients [2]. Finally, it would be useful to integrate our results on counter-models with the recently developed tableaux proof theory for Bunched Implications [4].

Acknowledgments

We would like to thank the anonymous referees for their valuable comments that suggested interesting improvements to the paper. Yang was supported by the US NSF under grant INT-9813854 and by Creative Research Initiatives of the Korean Ministry of Science and Technology. Calcagno and O'Hearn were supported by the EPSRC under the "Local Reasoning about State" projects.

References

1. M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for describing linked data structures. In *ESOP '99: European Symposium on Programming*, pages 2–19. Lecture Notes in Computer Science, Vol. 1576, S.D. Swierstra (ed.), Springer-Verlag, New York, NY, 1999.
2. W. Charatonik, S. Dal Zilio, A. Gordon, S. Mukhopadhyay, and J.M. Talbot. The complexity of model checking mobile ambients. In *FoSSaCS*, April 2001.
3. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995. ISBN 3-540-60149-X.
4. D. Galmiche and D. Méry. Proof-search and countermodel generation in propositional BI logic. In *TACS*, 2001. LNCS to appear.
5. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages*, January 2001.
6. J. Jenson, M. Jorgensen, N. Klarkund, and M. Schwartzback. Automatic verification of pointer programs using monadic second-order logic. In *Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation*, pages 225–236, 1997. SIGPLAN Notices 32(5).
7. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *Proceedings of 15th Annual Conference of the European Association for Computer Science Logic: CSL 2001*, pages 1–19. Springer-Verlag, 2001. LNCS 2142.
8. P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 99.
9. J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*. Palgrave, 2000.

Using Nondeterminism to Design Efficient Deterministic Algorithms

Jianer Chen[1,⋆], Donald K. Friesen[1], Weijia Jia[2,⋆⋆], and Iyad A. Kanj[3,⋆⋆⋆]

[1] Department of Computer Science, Texas A&M University,
College Station, TX 77843-3112.
{chen, friesen}@cs.tamu.edu
[2] Department of Computer Science, City University of Hong Kong,
Kowloon, Hong Kong SAR, China.
wjia@cs.cityu.edu.hk
[3] DePaul University, School of CTI,
243 S. Wabash Avenue, Chicago, IL 60604-2301.
ikanj@cs.depaul.edu, Fax: (312) 362-6116

Abstract. In this paper, we illustrate how nondeterminism can be used conveniently and effectively in designing efficient deterministic algorithms. In particular, our method gives an $O((5.7k)^k n)$ parameterized algorithm for the 3-D matching problem, which significantly improves the previous algorithm by Downey, Fellows, and Koblitz. The algorithm can be generalized to yield an improved algorithm for the r-D matching problem for any positive integer r. The method can also be employed in designing deterministic algorithms for other optimization problems as well.

1 Introduction

Nondeterminism has been a central topic in the study of complexity theory which lead to the famous "P $\stackrel{?}{=}$ NP" problem. In this paper, we study nondeterminism from an "algorithmic" point of view, and demonstrate how nondeterminism can be used conveniently and effectively to design efficient deterministic algorithms. We illustrate in detail our techniques by studying the complexity of the parameterized 3-D matching problem.

The 3-dimensional matching problem, abbreviated 3-D matching problem, is one of the six "basic" NP-complete problems according to Garey and Johnson [5]. Recently, Downey, Fellows, and Koblitz were the first to show that the r-D matching problem is fixed-parameter tractable [4]. They presented a parameterized algorithm for the problem based on families of hash functions. Their algorithm runs in time $O((rk)!(rk)^{3rk+1} n \log^6 n)$, where k is the size of the matching sought, and n the total number of tuples in the input set.

⋆ Supported in part by the National Science Foundation under Grants CCR-9613805, and CCR-0000206. Part of the work was done while this author was visiting City University of Hong Kong.
⋆⋆ Supported in part by UGC of Hong Kong under Grant 9040228.
⋆⋆⋆ Corresponding author.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 120–131, 2001.
© Springer-Verlag Berlin Heidelberg 2001

The authors of [4] noted that it is possible to improve the running time of their algorithm by using better families of hash functions. Using these families of hash functions, the running time of their algorithm can be improved to $O(2^{O(rk)}(rk)!n\log^4 n)$. In particular, for the 3-D matching problem, the running time of their algorithm can be improved to $O(2^{O(k)}(3k)!n\log^4 n)$.

We propose a completely different approach to develop an efficient deterministic parameterized algorithm for the 3-D matching problem. Our algorithm starts with a nondeterministic algorithm that solves the problem. The nondeterminism in the algorithm is then removed by a process called "de-nondetermination". With a careful design of the nondeterministic algorithm and with a nontrivial implementation of the de-nondetermination, we are able to show that the parameterized 3-D matching problem can be solved by a deterministic algorithm in time $O((5.7k)^k n)$. This is an improvement over the best algorithm in [4] by an enormous factor (greater than $k^{2k}\log^4 n$). Our algorithm can also be generalized to handle the r-D matching problem. More precisely, our method yields an algorithm of running time $O(\sqrt{r-1}(k^3+rn)(((r-1)^{r-1}k^{r-2})/(e^{r-2}(\sqrt[r]{2}-1)))^k)$ for the r-D matching problem. This is again a significant improvement over the best algorithm by a factor greater than $k^{2k}\log^4 n$. The method can also be employed in designing efficient algorithms for other optimization problems as well, such as packing and covering problems.

2 A Nondeterministic Algorithm

Let $S \subseteq X \times Y \times Z$ be a set of n (ordered) triples t_1, t_2, ..., t_n of symbols in $X \cup Y \cup Z$. Without loss of generality, we can assume that $|X| = |Y| = |Z| = n$ and that the symbol sets X, Y, and Z are pairwisely disjoint. Therefore, each symbol uniquely determines its own dimension. For a subset M of S, we sometimes also say, without ambiguity, that a symbol a is in M if the symbol a is contained in a triple in M.

A triple t_1 *conflicts* with a triple t_2 if t_1 and t_2 agree on any dimension but $t_1 \neq t_2$. A *matching* M in S is a subset of triples such that no two triples in M conflict with each other. The *parameterized 3-D matching* problem is to determine for a given pair (S, k) whether the set S of triples contains a matching of k triples. The problem is NP-complete if it is regarded as a general decision problem [5]. In the framework of fixed-parameter tractability theory [4], we assume that the parameter k is much smaller than the size n of the triple set S. Therefore, an algorithm solving the problem in time $O(f(k)n^c)$, where f is a function of the parameter k but independent of the input size n and c is a constant independent of the parameter k, is preferable to the straightforward algorithm of time $O(kn^k)$, which simply examines all subsets of size k in the triple set S.

In this section, we present a nondeterministic algorithm for the parameterized 3-D matching problem. We first assume that the triple set S has a matching M_k of k triples and discuss how such a matching can be found.

A *partial triple* t is a triple in which some of the symbols may be marked by an "unknown symbol $*$". Note that each triple in the set S is also a partial triple. A partial triple is *incomplete* if at least one of its symbols is a $*$ and a partial triple is a *complete triple* if it does not contain the symbol $*$. We first introduce the concepts of consistency and confliction of partial triples.

Definition 1. *Two partial triples $t = (a_1, a_2, a_3)$ and $t' = (a'_1, a'_2, a'_3)$ are consistent if for each i, $i = 1, 2, 3$, either $a_i = a'_i$ or one of a_i and a'_i (or both) is $*$. Two partial triples conflict with each other if they are not consistent but agree on a dimension.*

A set P of partial triples is called a *partial matching* if there is a one-to-one mapping from P to a matching M in S such that each partial triple in P is consistent with the corresponding triple in M. In this case, we say that the partial matching P is *consistent* with the matching M. According to the definition, a partial matching can always be obtained from a consistent matching in S by replacing certain symbols by the symbol $*$.

The basic idea of our nondeterministic algorithm is to start with a partial matching P_k, which consists of k partial triples of the form $(*, *, *)$, then to decide which symbol should replace each $*$ symbol in the partial matching, thus obtaining a matching of k triples in S. We first show how to obtain the first non-$*$ symbol for each partial triple in P_k.

A matching M_0 in S is *maximal* if every triple in $S - M_0$ conflicts with at least one triple in M_0. A maximal matching M_0 in S can be easily constructed in linear time as follows. Start from an empty matching M_0 and mark each symbol as "unused". Now go through the set S. For each triple t in S in which no symbol is used, add t to the matching M_0 and mark the symbols in t as "used". The resulting matching M_0 is clearly a maximal matching in S. We shall see in the discussion below how maximal matchings prove to be very helpful in constructing a matching M_k of k triples. First we prove the following lemma.

Lemma 1. *Let M_0 be a maximal matching in S and M_k be a matching of k triples in S. Then every triple in M_k has at least one symbol in M_0.*

Proof. If a triple t in M_k has no symbol contained in M_0, then t does not conflict with any triples in M_0. This contradicts the assumption that M_0 is a maximal matching in S. □

We call $(M_k, \{a_1, \ldots, a_k\})$ a *feasible pair* if M_k is a matching of k triples in S, and $\{a_1, \ldots, a_k\}$ is a set of symbols of S, such that each triple in M_k contains exactly one symbol in the set $\{a_1, \ldots, a_k\}$.

Lemma 2. *Suppose that the triple set S has a matching of k triples. Let M_0 be a maximal matching of less than k triples in S. Then there is a feasible pair $(M_k, \{a_1, \ldots, a_k\})$ such that all symbols in $\{a_1, \ldots, a_k\}$ are in M_0 and each triple in M_0 contains at least one symbol in the set $\{a_1, \ldots, a_k\}$.*

Proof. Suppose the maximal matching M_0 contains $k' < k$ triples. Let $M_{\max}$ be a *maximum* matching in the triple set S. Since S has a matching of k triples, the matching $M_{\max}$ contains at least k triples. Construct a bipartite graph $G = (V_1 \cup V_2, E)$ as follows. Each triple in the maximal matching M_0 is a vertex in V_1 and each triple in the maximum matching $M_{\max}$ is a vertex in V_2 (thus if a triple t is in both M_0 and $M_{\max}$, then t makes two vertices in G). There is an edge in E from a vertex t in V_1 to a vertex t' in V_2 if the triple t in M_0 and the triple t' in $M_{\max}$ have a symbol in common. Let Q be a maximum graph matching in the graph G.

We first show that the maximum graph matching Q contains k' edges. In fact, if Q contains less than k' edges, then by Hall's Theorem (see for example [1], page 72, Theorem 5.2), there is a subset V' in V_1 such that the number of vertices in the set $N(V')$ is strictly smaller than the number of vertices in V', where $N(V')$ is the set of vertices in V_2 that are adjacent to any vertex in V'. Now since the vertices in V' are not adjacent to any vertices in $V_2 - N(V')$, the set $V_2 - N(V') + V'$ would form a matching in S that contains more triples than the matching $M_{\max}$. This contradicts the assumption that $M_{\max}$ is a maximum matching in S.

Therefore, the maximum graph matching Q contains k' edges. Let t_1, ..., $t_{k'}$ be the k' vertices in V_1 and t'_1, ..., $t'_{k'}$ be the k' vertices in V_2 such that (t_i, t'_i) is an edge in Q for $i = 1, \ldots, k'$. Let a_i be a common symbol in the triples t_i and t'_i for $i = 1, \ldots, k'$. Pick any $k - k'$ triples $t'_{k'+1}$, ..., t'_k from $M_{\max} - \{t'_1, \ldots, t'_{k'}\}$ (recall that $M_{\max}$ contains at least k triples). The k triples t'_1, ..., $t'_{k'}$, $t'_{k'+1}$, ..., t'_k make a matching M_k of k triples in S. By Lemma 1, each of the triples t'_j, $j = k'+1, \ldots, k$ contains a symbol a_j in the maximal matching M_0. Now $(M_k, \{a_1, \ldots, a_{k'}, a_{k'+1}, \ldots, a_k\})$ is a feasible pair where all symbols in $\{a_1, \ldots, a_{k'}, a_{k'+1}, \ldots, a_k\}$ are in the maximal matching M_0, and each triple t_i in $M_0 = \{t_1, \ldots, t_{k'}\}$ contains the symbol a_i in the set $\{a_1, \ldots, a_{k'}, a_{k'+1}, \ldots, a_k\}$, for $i = 1, \ldots, k'$. □

Call a set Y_k of k symbols in M_0 a *spanning set* if each triple in M_0 contains at least one symbol in Y_k. According to Lemma 2, if S contains a matching M_k of k triples, then there exists a spanning set Y_k of M_0 such that (M_k, Y_k) is a feasible pair. Now here is the first step to construct the matching M_k of k triples in S. First we construct a maximal matching M_0 in S. Suppose that the maximal matching M_0 contains k' triples. By our assumption, S has a matching M_k of k triples. Thus, we must have $k' \geq k/3$ (see, for example, [8] for a proof of this simple fact). If $k' \geq k$, then we simply pick any k triples in M_0 to make a matching of k triples. On the other hand, if $k/3 \leq k' < k$, then according to Lemma 2, there is a partial matching P_k of k partial triples, consistent with the matching M_k, and a spanning set Y_k of M_0, such that each partial triple in P_k contains one symbol in Y_k and two $*$'s. To construct the partial matching P_k, we start with a set of k empty triples (i.e., triples containing $*$ symbols only), then we nondeterministically guess a spanning set Y_k of M_0, and for every triple in P_k we replace a $*$ symbol with a disticnt symbol in Y_k. (note that each guessed

symbol uniquely determines its own dimension in a partial triple in P_k). We will call the partial matching P_k the *starting partial matching*.

Beginning with the starting partial matching, inductively we assume that we have obtained a partial matching P_k of k partial triples in which each partial triple contains at most two $*$'s. In order to construct the matching M_k by filling in the unknown symbols $*$ in the partial matching P_k, we try to replace each incomplete partial triple t' in P_k by a consistent triple t in the set S. We say that the partial triple t' in P_k is *replaceable* by a triple t in S if t does not conflict with any partial triple in P_k and t' is the unique partial triple in P_k that is consistent with t. We repeat this process of substituting replaceable partial triples in P_k by triples in S until there are no more replaceable partial triples in P_k, using the greedy algorithm GREEDY-FILLING given in Figure 1.

Algorithm. GREEDY-FILLING
INPUT: a partial matching P_k of k partial triples
OUTPUT: a matching M in S

1. $Q_k = P_k$;
2. **for** each triple t in the set S **do**
   if a partial triple t' in Q_k is replaceable by t
   then $Q_k = Q_k - \{t'\} + \{t\}$;
3. let M be the set of complete triples in Q_k

Fig. 1. The algorithm GREEDY-FILLING

The running time of the algorithm GREEDY-FILLING can be bounded by $O(n)$ if for each symbol, we keep a mark as "unused" or "used", and for each non-$*$ symbol in Q_k, we also attach to it the other non-$*$ symbols in the same partial triple in Q_k.

There are two possible cases for the matching M constructed by the algorithm GREEDY-FILLING: either M contains k triples in S or M contains fewer than k triples. The first case is simple: we have found a matching of k triples in S so we are done. Now we consider the second case. Suppose that the matching M consists of $k' < k$ triples. First note that if t is a complete triple in P_k, then t must be contained in the constructed matching M.

By our assumption, the partial matching P_k is consistent with the matching M_k. According to the algorithm GREEDY-FILLING, the matching M is obtained from a subset P'_k of k' partial triples in P_k by replacing the $*$ symbols in P'_k by proper symbols. Thus, the matching M is consistent with the partial matching P'_k. We say a symbol is a *newly added symbol* if it is in the matching M but not in the partial matching P_k. Note that the subset P'_k of P_k can be easily identified from the matching M. Now since the partial matching P_k is consistent with the matching M_k, the partial matching P'_k in P_k is also consistent with a subset M'_k of k' triples in the matching M_k. Since $k' < k$, the set $M_k - M'_k$ is not empty. Let t be any triple in the set $M_k - M'_k$. The triple t is consistent with a partial triple t' in $P_k - P'_k$. Thus, the triple t is not contained in the matching M. By

the algorithm GREEDY-FILLING, the triple t conflicts with at least one triple t'' in M. Suppose that the triples t and t'' share the same symbol a.

We claim that the symbol a is a newly added symbol in the matching M. In fact, if a is in the partial matching P_k, then since a is in the matching M which is consistent with the partial matching P'_k, the symbol a is also in P'_k. In consequence, the symbol a is contained in a triple in the subset M'_k. This contradicts the assumption that M_k is a matching since the triple t in the set $M_k - M'_k$ also contains the symbol a. This proves that the symbol a is a newly added symbol in the matching M. By this reasoning, for each partial triple t' in $P_k - P'_k$, which is consistent with a triple t in $M_k - M'_k$, one of the symbols in t that corresponds to a $*$ in t' is contained in the matching M. Moreover, this symbol must be a newly added symbol in M. Therefore, to decide which symbol should replace a $*$ in t', we nondeterministically guess a newly added symbol in M and replace the $*$ in t' by this guessed symbol. Note that this process has reduced the number of $*$ symbols in P_k by 1. Now on this new partial matching P_k with one fewer $*$, we apply the algorithm GREEDY-FILLING again. By repeating this process at most $2k$ times, each time either we construct a matching of k triples in S or reduce the number of $*$ symbols in P_k by 1, we can eliminate all $*$ symbols in the partial matching P_k, thus obtaining a matching of k triples in S.

Can we do better than iterating this process $2k$ times? We show next that it suffices to iterate this process k times. We start by showing that if each partial triple in P_k contains at most one $*$ symbol, then the matching M_k can be constructed easily.

Lemma 3. *Suppose that P is a partial matching of h partial triples such that each partial triple in P contains at most one $*$. Then a (regular) matching in S consistent with the partial matching P can be constructed in time $O(h^3 + n)$.*

Proof. By the definition of partial matchings, there is a one-to-one mapping from the partial matching P to a matching M of h triples in S such that each partial triple in P is consistent with the corresponding triple in M. Let $P = P_0 \cup P_1$, where P_0 is the set of partial triples in P that contain no $*$ symbols and P_1 is the set of partial triples in P that contain exactly one $*$ symbol. By the definition, the matching M can also be partitioned into $M = M_0 \cup M_1$, where $M_0 = P_0$ and M_1 is a matching consistent with the partial matching P_1. Suppose that the partial matching P_1 consists of $h' \leq h$ partial triples. We only need to show how a matching M' of h' triples can be constructed from the partial matching P_1 such that $M' \cup P_0$ is a matching of h triples in S.

Construct a bipartite graph G from the partial matching P_1 as follows. The graph G contains two kinds of vertices: each partial triple in P_1 is a vertex in G, and each non-$*$ symbol a not in P is a vertex in G if there is a partial triple t in P_1 such that the two non-$*$ symbols in t and the symbol a make a triple in S. The vertices in G corresponding to partial triples in P_1 will be called *triple-vertices* and the vertices in G corresponding to symbols will be called *symbol-vertices*. There is an edge in G from a triple-vertex t in P_1 to a symbol-vertex a if the

two non-$*$ symbols in t and the symbol a make a triple in S. Note that there are exactly h' triple-vertices.

The graph G has a graph matching of h' edges: for example the matching M_1 of h' triples in S consistent with the partial matching P_1 corresponds to a graph matching of h' edges in G. Moreover, every maximum graph matching, which consists of exactly h' edges, corresponds to a matching M' of h' triples in S that is consistent with the partial matching P_1 such that $M' \cup P_0$ is a matching of h triples in S (note that all symbol-vertices of the graph G correspond to symbols not in P). Therefore, a matching of h triples in S can be constructed by constructing a maximum graph matching in the graph G.

If a triple-vertex t in the graph G is incident on more than h' edges, then we can first construct a maximum graph matching (of $h'-1$ edges) in the graph $G - \{t\}$ then add one more edge incident to t to make a graph matching of h' edges in the graph G. Therefore, after a time $O(n)$ preprocessing, we can assume, without loss of generality, that each triple-vertex in the graph G is incident on at most h' edges. Consequently, the graph G contains at most $(h')^2$ edges, and at most $(h')^2 + h'$ vertices (h' triple-vertices and at most $(h')^2$ symbol-vertices). Using Hopcroft and Karp's matching algorithm for bipartite graphs [7], which runs in time $O(m\sqrt{n})$ on graphs of n vertices and m edges, we conclude that a maximum graph matching in G can be constructed in time $O((h')^3) = O(h^3)$. As a result, a matching of h triples in the triple set S can be constructed in time $O(h^3 + n)$. □

Therefore, beginning with the starting partial matching P_k, in which each partial triple contains exactly two $*$'s, we only need to replace one $*$ symbol in each partial triple in P_k. Once every partial triple in the partial matching P_k contains exactly one $*$ symbol, we can apply Lemma 3 to construct directly a matching of k triples in S. This method will reduce the number of guessed symbols from $2k$ to k.

To make this method possible, we do the following. Given the partial matching P_k of k partial triples, let T_1 be the set of h partial triples in P_k that contain at most one $*$ symbol and let T_2 be the set of h' triples in P_k that contain two $*$ symbols, $k = h + h'$. Now instead of applying the subroutine Greedy-Filling on P_k directly to construct the matching M, we first construct a matching M_1 of h triples from the partial matching T_1, then apply the subroutine Greedy-Filling on the partial matching T_2 to construct a matching M_2 of at most h' triples such that $M = M_1 \cup M_2$ is a matching. Now note that every partial triple t in P_k that is not consistent with a triple in M must be a triple in T_2, which has two $*$ symbols. Thus, guessing a newly added symbol in M to replace a $*$ symbol in t will make t a triple with exactly one $*$ symbol. Using this method, we can avoid guessing any symbols for the third $*$ symbol in any partial triple in P_k.

We summarize all the above discussions in the algorithm given in Figure 2.

Theorem 1. *The algorithm* Nondet-3D Matching *solves the parameterized 3-D matching problem if it makes all correct guesses.*

Algorithm. NONDET-3D MATCHING
INPUT: the set S of n triples and an integer k

1. construct a maximal matching M_0 in S;
2. **case 1.** $|M_0| \geq k$
   output any k triples in M_0; stop.
3. **case 2.** $|M_0| < k/3$
   stop: S does not contain a matching of k triples.
4. **case 3.** $k/3 \leq |M_0| < k$
   for each spanning set Y_k of k symbols in M_0
   4.1. construct from Y_k a partial matching P_k of k partial triples, each containing exactly two $*$'s;
   4.2. **loop** k times
   let $P_k = T_1 \cup T_2$, where T_1 is the set of h partial triple s in P_k that contain at most one $*$, and T_2 is the set of h' partial triples in P_k that contain exactly two $*$'s, $k = h + h'$;
   construct a matching M_1 of h triples consistent with T_1;
   let S' be the triple set obtained from S by deleting triples that contain symbols in M_1;
   call GREEDY-FILLING to construct a matching M_2 from T_2 and S';
   $M = M_1 \cup M_2$;
   if $|M| = k$
   then output M; stop.
   else
   if M contains no newly added symbols
   then stop: S does not contain a matching of k triples
   else
   pick any triple t in P_k that is not consistent with a triple in M;
   guess a newly added symbol in M to replace a $*$ in t
   4.3. stop: S does not contain a matching of k triples.

Fig. 2. The nondeterministic algorithm for 3-D matching

Proof. According to the above analysis, if the triple set S contains a matching M_k of k triples and if the algorithm makes all its guesses correctly, then the algorithm will eventually end up with the matching M_k of k triples in S.

On the other hand, suppose that the triple set S has no matching of k triples. Then since the algorithm NONDET-3D MATCHING concludes the existence of a matching of k triples only when it actually constructs such a matching, the algorithm will never make an incorrect conclusion if such a matching does not exist. In fact, in this case either the algorithm finds out that the maximal matching M_0 constructed in step 1 contains less than $k/3$ triples, or finds out in step 4.2 that no new symbols can be added to the partial matching P_k. Once either of these conditions is realized, the algorithm correctly concludes the nonexistence of the matching of k triples in S. □

3 De-nondetermination

The *de-nondetermination* process is to remove nondeterminism in a given nondeterministic algorithm and convert it into a deterministic algorithm. One way to do this is to count the number of binary bits guessed in the algorithm, then enumerate and check all possible combinations of these binary bits. However, this blind enumeration approach might be costly. In the following we show that a more careful implementation of the de-nondetermination process can result in a more efficient deterministic algorithm for the problem than the one resulting from blind enumeration. We adopt two techniques for efficient de-nondeterminations: the first one is to use more careful enumeration of valid situations instead of simply counting the number of guessed binary bits, and the second is to perform more thorough combinatorial analysis to reduce the search space so that fewer guessed binary bits will be needed.

First consider the computational complexity of the algorithm NONDET-3D MATCHING. Suppose that the triple set S has n triples. As explained in section 2, the maximal matching M_0 can be constructed in time $O(n)$, and each execution of the algorithm GREEDY-FILLING takes time $O(n)$. Since the loop in step 4.2. is executed at most k times, we conclude that the running time of the nondeterministic algorithm NONDET-3D MATCHING is bounded by $O(kn)$. We analyse next the deterministic running time of the algorithm.

Since steps 1, 2, and 3 of the algorithm NONDET-3D MATCHING take $O(n)$ deterministic time, it suffices to analyse the time taken to de-nondeterminize step 4 of the algorithm. Consider step 4.1., we count how many spanning sets there are and how they can be enumerated. Let $D(k', k)$ be the number of spanning sets of k symbols in the matching M_0 of k' triples. The function $D(k', k)$ satisfies the following recurrence relation:

$$D(k', k) = 3D(k' - 1, k - 1) + 3D(k' - 1, k - 2) + D(k' - 1, k - 3) \qquad (1)$$

with the boundary conditions $D(k', k) = 0$ for $k' > k$ or $k' < k/3$. This recurrence relation can be seen as follows. Fix any triple t in M_0 and let Y_k be a spanning set of k symbols in M_0. Suppose that the triple t contains exactly one symbol a in the spanning set Y_k, then (recursively) the matching $M_0 - \{t\}$ has $D(k' - 1, k - 1)$ spanning sets of $k - 1$ symbols, each of which together with the symbol a makes a spanning set for M_0. Moreover, there are three different ways (one for each dimension) for the triple t to contain exactly one symbol in a spanning set. Therefore, totally there are $3D(k' - 1, k - 1)$ spanning sets of k symbols in M_0 such that the triple t contains exactly one symbol in the spanning sets. Similarly, there are $3D(k' - 1, k - 2)$ spanning sets of k symbols in M_0 such that the triple t contains exactly two symbols in the spanning sets, and there are $D(k' - 1, k - 3)$ spanning sets of k symbols in M_0 such that the triple t contains exactly three symbols in the spanning sets. This gives the recurrence relation in (1).

It is easy to verify that $D(k', k) \leq \alpha_0^k$, where $\alpha_0 = 3.8473\cdots$ is the positive root for the equation $x^3 - 3x^2 - 3x - 1 = 0$. Moreover, the spanning sets of k symbols in M_0 can be systematically and recursively enumerated according to the

recurrence relation (1). We conclude that step 4.1 of the algorithm NONDET-3D MATCHING can be implemented by a deterministic loop whose body is executed at most $\alpha_0^k \leq 3.85^k$ times.

Now we consider step 4.2., in the first execution of the loop in step 4.2, the partial matching P_k has exactly $2k$ $*$ symbols. Therefore, the number of newly added symbols in the matching M is at most $2(k-1)$ (note the matching M has less than k triples and M is consistent with a subset of P_k). Thus, there are at most $2(k-1)$ ways to pick a symbol in M to replace a $*$ in a triple t in P_k. In general, in the ith execution of the loop in step 4.2, the partial matching P_k contains $2k-i+1$ $*$ symbols, and the number of newly added symbols in the matching M is at most $2(k-1)-i+1$ (again note that the matching M has less than k triples and any triple in P_k that is not consistent with a triple in M has two $*$ symbols). Thus, in the ith execution of the loop in step 4.2, we have at most $2(k-1)-i+1$ different ways to pick a newly added symbol in M to replace a $*$ symbol in a triple in P_k that is not consistent with any triple in M. Based on the above discussion, the loop in step 4.2 can be implemented as following. For each execution of the loop in step 4.2, we start with a sequence of positive integers

$$d_1 d_2 \cdots d_k \qquad 1 \leq d_i \leq 2(k-1)-i+1, \text{ for } 1 \leq i \leq k \tag{2}$$

such that in the ith execution of the loop body in step 4.2, we pick the d_ith newly added symbol in the matching M (if the matching M has less than d_i newly added symbols, we simply skip this sequence). The de-nondetermination of the last line in step 4.2 can be implemented by executing the loop in step 4.2 based on every possible sequence of positive integers of form (2).

Therefore, step 4.2 of the algorithm NONDET-3D MATCHING can be de-nondeterminated into a deterministic step by a **for** loop on all possible sequences of positive integers of form (2). On each execution of the **for** loop, the loop in step 4.2 is executed. In consequence, the loop body in step 4.2 of the algorithm is executed at most

$$k \cdot [2(k-1)] \cdot [2(k-1)-1] \cdots [2(k-1)-k+1] = \frac{k(2k-2)!}{(k-2)!}$$

times. Within each execution of the loop body, the sets T_1 and T_2 can be trivially constructed from the partial matching P_k in time $O(k)$. According to Lemma 3, the matching M_1 of h triples can be constructed from T_1 in time $O(h^3+n) = O(k^3+n)$. The subroutine GREEDY-FILLING takes time $O(n)$. In conclusion, in the corresponding deterministic algorithm, each execution of the step 4.2 takes time bounded by

$$O((k^3+n) \cdot \frac{k(2k-2)!}{(k-2)!}) = O(\frac{(k^4+kn)(2k-2)!}{(k-2)!})$$

Again using Stirling's approximation, we get

$$\frac{(2k-2)!}{(k-2)!} \leq \frac{(2k)!}{k!} \leq 2(4k/e)^k$$

where $e = \sum_{i \geq 0}(1/i!) = 2.7182818\cdots$ is the natural logarithmic base.

Combining this with the analysis of the main **for** loop in step 4, we conclude that the running time of the deterministic algorithm, obtained from the denondetermination of the algorithm NONDET-3D MATCHING, is bounded by

$$O(3.85^k(k^4+kn)(4k/e)^k) = O((k^4+kn)(15.4k/e)^k) = O((5.7k)^k n)$$

The last equality is obtained from the fact that $15.4/e < 5.67$ so that we have both $k^4(15.4k/e)^k$ and $k(15.4k/e)^k$ are of order $O((5.7k)^k)$. We conclude with the following theorem:

Theorem 2. *The parameterized 3-D matching problem can be solved by a deterministic algorithm of running time bounded by* $O((5.7k)^k n)$.

4 Generalization and Concluding Remarks

The algorithm in Figure 2 for the 3-D matching problem can be generalized in a straightforward manner to give an algorithm for the general r-D matching problem. We have the following theorem:

Theorem 3. *The parameterized r-D matching problem can be solved by a deterministic algorithm of running time* $O(\sqrt{r-1}(k^3 + rn)(((r-1)^{r-1}k^{r-2})/(e^{r-2}(\sqrt[r]{2}-1)))^k)$.

The reader is refered to [3] for the detailed algorithm and its proof of correctness.

We have demonstrated, using the parameterized 3-D matching and r-D matching problems as examples, how nondeterminism can be used to develop efficient deterministic algorithms. Our algorithm improves the previous best algorithm by Downey, Fellows, and Koblitz [4], by a factor greater than $k^{2k}\log^4 n$. In particular, our algorithm for the 3-D matching problem has running time $O((5.7k)^k n)$, which is significantly faster than the algorithm in [4] of running time $O(2^{O(k)}(1.1k)^{3k} n \log^4 n)$. The techniques presented in the current paper are also applicable to other parameterized optimization problems as well. For example, it is not hard to see that the algorithms presented in this paper can be modified to give algorithms for packing problems, such as the r-D packing problem [5]. Moreover, these techniques can be adopted in solving covering problems, like the vertex cover problem [2], which leads to a simpler and shorter presentation of the solutions.

One might argue that the techniques presented here are essentially exhaustive search techniques, like the standard techniques used in parameterized algorithms [4], for which we could not disagree. However, we must indicate that conceptually our techniques provide a more natural and intuitive method for developing efficient deterministic algorithms for certain problems where applying direct exhaustive search may be lengthy and confusing.

References

1. J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, New York, 1976.
2. J. Chen, I. A. Kanj, and W. Jia, "Vertex Cover: further observations and further improvements," in *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, *Lecture Notes in Computer Science*, vol. 1665, pp. 313-324, Ascona, Switzerland, June 1999.
3. J. Chen, D. K. Friesen, W. Jia, and I. A. Kanj, "Using nondeterminism to design efficient deterministic algorithms," *Technical Report*, Department of Computer Science, Texas A&M University, College Station, Texas, May 2001.
4. R. G. Downey and M. R. Fellows, *Parameterized Complexity*, New York, New York: Springer, (1999), pp. 220-222.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
6. R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, 1994.
7. J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, *SIAM J. Comput. 2*, (1973), pp. 225-231.
8. V. Kann, Maximum bounded 3-dimensional matching is MAX SNP-complete, *Information Processing Letters 37*, (1991), pp. 27-35.

Liveness Verification of Reversal-Bounded Multicounter Machines with a Free Counter ⋆

(Extended Abstract)

Zhe Dang[1], Oscar H. Ibarra[2], and Pierluigi San Pietro[3]

[1] School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164
[2] Department of Computer Science
University of California
Santa Barbara, CA 93106
[3] Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italia

Abstract. We investigate the Presburger liveness problems for nondeterministic reversal-bounded multicounter machines with a free counter (NCMFs). We show the following:

- The $\exists$-Presburger-i.o. problem and the $\exists$-Presburger-eventual problem are both decidable. So are their duals, the $\forall$-Presburger-almost-always problem and the $\forall$-Presburger-always problem.
- The $\forall$-Presburger-i.o. problem and the $\forall$-Presburger-eventual problem are both undecidable. So are their duals, the $\exists$-Presburger-almost-always problem and the $\exists$-Presburger-always problem.

These results can be used to formulate a weak form of Presburger linear temporal logic and develop its model-checking theories for NCMFs. They can also be combined with [12] to study the same set of liveness problems on an extended form of discrete timed automata containing, besides clocks, a number of reversal-bounded counters and a free counter.

1 Introduction

An infinite-state system can be obtained by augmenting a finite automaton with one or more unbounded storage devices. The devices can be, for instance, counters (unary stacks), pushdown stacks, queues, and/or Turing tapes. However, an infinite-state system can easily achieve Turing-completeness, e.g., when two counters are attached to a finite automaton (resulting in a "Minsky machine"). For these systems, even simple problems such as membership are undecidable.

In the area of model-checking, the search for (efficient) techniques for verifying infinite-state systems has been an ongoing research effort. Much work has been devoted to investigating various restricted models of infinite-state systems that are amenable to

⋆ Supported in part by NSF grant IRI-9700370

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 132–143, 2001.
© Springer-Verlag Berlin Heidelberg 2001

automatic verification. The work is motivated by the successes of "efficient" model-checking techniques for finite-state systems such as hardware devices and reactive systems [20], and the need for developing practical techniques for deciding verification properties of infinite-state systems.

The infinite-state models that have been investigated include timed automata [1], pushdown automata [3,14], various versions of counter machines [5,13,18], and various queue machines [2,4,16,17,21].

Counter machines are considered a natural model for specifying reactive systems containing integer variables. They have also been found to have a close relationship to other popular models of infinite-state systems, such as timed automata [1]. In [6], it was shown that, as far as binary reachability (the set of configuration pairs such that one can reach the other) is concerned, a timed automaton can be transformed into a particular type of counter machine without nested cycles [5]. In contrast to [6], timed automata (with discrete time) are mapped to counter machines with reversal-bounded counters in [8]. In the case of dense time, the same mapping applies using some pattern technique [7].

Thus, studying various restricted models of counter machines may help researchers to develop verification theories concerning infinite-state systems such as timed automata augmented with unbounded storage [8].

In this paper, we focus on a class of restricted counter machines, called nondeterministic reversal-bounded multicounter machines with a free counter (NCMFs). More precisely, an NCMF M is a nondeterministic finite automaton augmented with a finite number of reversal-bounded counters (thus, in any computation, each counter can change mode from nondecreasing to nonincreasing and vice-versa at most r times for some given nonnegative integer r) and one free counter (which need not be reversal-bounded). A fundamental result is that the emptiness problem for languages accepted by NCMFs is decidable [15]. But here we do not use NCMFs as language recognizers; instead, we are interested in the behaviors they generate. So, unless otherwise specified, an NCMF has no input tape. Reversal-bounded counters are useful in verification of reactive systems. For instance, a reversal-bounded counter can be used to count the number of times a particular external event occurs in a reactive system – in this case, the counter is simply 0-reversal-bounded, i.e., non-decreasing. Allowing a free counter, together with other reversal-bounded counters, makes the reactive system infinite-state. More application issues of NCMFs and the results in this paper can be found at the end of the paper.

The study of safety properties and liveness properties of infinite-state systems is of great importance in the area of formal verification. Safety properties look at only finite (execution) paths; mostly they can be reduced to reachability problems. In [18], it was shown that the Presburger safety analysis problem is decidable for NCMFs and their generalizations. A typical example of a Presburger safety property that we might want to verify for an NCMF M with counters x_1, x_2 and x_3 is the following: Starting from counter values satisfying $x_1 - x_2 + 3x_3 > 5$, M can only reach counter values satisfying $x_1 + 2x_2 - 4x_3 < 8$.

In this paper, we systematically study a number of Presburger liveness problems for NCMFs. An example is a $\exists$-Presburger-i.o. problem like: Given an NCMF M with counters x_1, x_2 and x_3, does there exist an ω-path (i.e., infinite execution path) p for

M such that $x_1 + 2x_2 - 4x_3 < 8$ is satisfied on p infinitely often? The research presented in this paper is inspired by the recent work in [12] that investigates the same set of Presburger liveness problems for discrete timed automata. But the techniques we develop here are completely different from the ones in [12]. Clocks in a discrete timed automaton, when considered as counters, are synchronous. So, in some way, a discrete timed automaton can be treated as a reversal-bounded multicounter machine (an NCMF without the free counter) [8]. The ability of an NCMF to use a free counter makes the Presburger liveness proofs much more complicated. The main results of this paper show that the $\exists$-Presburger-i.o. problem is decidable for NCMFs. This result leads us to conjecture that the $\exists$-Presburger-i.o. problem is also decidable for (discrete timed) pushdown processes when the counts on individual stack symbols are part of the Presburger property being verified [8].

The paper is organized as follows. Section 2 introduces the main definitions. Section 3 shows the decidability of the $\exists$-Presburger-i.o. and $\exists$-Presburger-eventual problems. Section 4 generalizes the proofs in [12] to show the undecidability of the $\forall$-Presburger-i.o. and the $\forall$-Presburger-eventual problems. Section 5 is a conclusion.

2 Preliminaries

Let $X = \{x_0, \cdots, x_k\}$ be a finite set of integer variables. A formula $\Sigma_{0 \le i \le k} a_i x_i \# b$, where a_i and b are integers, is called an *atomic linear constraint*, if $\#$ is $>$ or $=$. The formula is called an *atomic mod-constraint*, if $\#$ is $\equiv_d$ for some $d > 0$. A *linear-conjunction* is a conjunction of a finite number of atomic linear constraints. A *linear-mod-conjunction* is a conjunction of a finite number of atomic linear constraints and atomic mod-constraints. It is well known that a Presburger formula [19] (first-order formula over integers with addition) can always be written as a disjunctive normal form of atomic linear constraints and atomic mod-constraints, i.e., a disjunction of linear-mod-conjunctions. A set P is *Presburger-definable* if there exists a Presburger formula F on X such that P is exactly the set of the solutions for X that make F true. It is well known that the class of Presburger-definable sets does not change if quantifications are allowed. Hence, when considering Presburger formulas, we will allow quantifiers over integer variables. A *standard test* on X is a Boolean combination of *atomic tests* in the form of $x \# c$, where $\#$ denotes $\le, \ge, <, >$, or $=$, c is an integer, $x \in X$. Let $\mathcal{T}_X$ be the set of all standard tests on X.

A nondeterministic multicounter machine (NCM) M is a nondeterministic machine with a finite set of (control) states and a finite number of integer counters. Each counter can add 1, subtract 1, or stay unchanged. M can also test whether a counter is equal to, greater than, or less than an integer constant by performing a standard test. Without loss of generality, in this paper we consider M without event labels on transitions, since these labels can be built into the control states.

Formally, a *nondeterministic multicounter machine (NCM)* M is a tuple $\langle S, X, E \rangle$ where S is a finite set of *(control) states*, X is a finite set of integer counters, and $E \subseteq S \times \mathcal{T}_X \times \{-1, 0, 1\}^{|X|} \times S$ is a finite set of *edges* or *transitions*. Each edge $\langle s, t, \textbf{incr}, s' \rangle$ denotes a transition from state s to state s' with $t \in \mathcal{T}_X$ being the *test* or

the enabling condition. **incr** $\in \{-1, 0, 1\}^{|X|}$ denotes the effect of the edge: each counter in X is incremented by the amount specified in vector **incr**.

The semantics of NCMs is defined as follows. We use $\boldsymbol{V}$ to denote counter vectors (i.e., vectors of counter values). We use $\boldsymbol{V}_i$ to denote the value of counter x_i in $\boldsymbol{V}$, for $0 \leq i \leq |X|$. A *configuration* $\langle s, \boldsymbol{V}\rangle \in S \times \mathbf{Z}^{|X|}$ is a pair of a control state s and a counter vector $\boldsymbol{V}$. $\langle s, \boldsymbol{V}\rangle \rightarrow_M \langle s', \boldsymbol{V}'\rangle$ denotes a *one-step transition* from configuration $\langle s, \boldsymbol{V}\rangle$ to configuration $\langle s', \boldsymbol{V}'\rangle$ satisfying the following conditions:

- There is an edge $\langle s, t, \textbf{incr}, s'\rangle$ in M connecting state s to state s',
- The enabling condition of the edge is satisfied, that is, $t(\boldsymbol{V})$ is true,
- Each counter changes according to the edge, i.e., $\boldsymbol{V}' = \boldsymbol{V} + \textbf{incr}$.

A *path* is a finite sequence

$$\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle$$

such that $\langle s_i, \boldsymbol{V}^i\rangle \rightarrow_M \langle s_{i+1}, \boldsymbol{V}^{i+1}\rangle$ for each $0 \leq i \leq n-1$. An *ω-path* is an infinite sequence $\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle \cdots$ such that each prefix $\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle$ is a path. We write $\langle s, \boldsymbol{V}\rangle \rightsquigarrow_M \langle s', \boldsymbol{V}'\rangle$ if the configuration $\langle s, \boldsymbol{V}\rangle$ reaches the configuration $\langle s', \boldsymbol{V}'\rangle$ through a path in M. The binary relation $\rightsquigarrow_M$ is called *binary reachability*.

It is well known that counter machines with two counters have an undecidable halting problem. Thus, in order to investigate any nontrivial decidable verification problems for NCMs, we have to restrict the behaviors of the counters. A counter is *r-reversal-bounded* if it changes mode between nondecreasing and nonincreasing at most r times. For instance, the following sequence of counter values:

$$0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 3, 2, 1, 1, 1, 1, \cdots$$

exhibits only one counter reversal. M is *reversal-bounded* if each counter in M is r-reversal-bounded for some r. M is a *reversal-bounded NCM with a free counter* (NCMF) if M has a number of reversal-bounded counters and an unrestricted counter (that need not be reversal-bounded). ¿From now on, an NCM (NCMF) refers to a machine with reversal-bounded counters (and one free counter). We assume throughout that whenever we are given an NCM (NCMF), the reversal-bound r is also specified.

A fundamental result for NCMFs is that the binary reachability is Presburger. This characterization is quite useful, since it is well known that the emptiness and the validity problems for Presburger formulas are decidable.

Theorem 1. *The binary reachability is effectively Presburger definable for a reversal-bounded nondeterministic multicounter machine with a free counter. [15,18]*

This fundamental result allows us to automatically verify a *Presburger safety analysis problem* for an NCMF M [8,18]: from configurations in I, M can only reach configurations in P, where I and P are Presburger definable sets of configurations. This problem is equivalent to $\neg\exists\alpha\exists\beta(\alpha \in I \wedge \alpha \rightsquigarrow_M \beta \wedge \beta \in \neg P)$, which, from Theorem 1, is Presburger and therefore decidable.

In this paper, we systematically investigate *Presburger liveness analysis problems* for NCMFs by considering their ω-paths. We follow the notations in [12]. Let M be an NCMF, I and P be two Presburger-definable sets of configurations, and p be an ω-path $\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle \cdots$. We say that p *starts* from I if $\langle s_0, \boldsymbol{V}^0\rangle \in I$. Define

- p is *P-i.o.* if P is satisfied infinitely often on the ω-path, i.e., there are infinitely many n such that $\langle s_n, \boldsymbol{V}^n\rangle \in P$.
- p is *P-always* if for each n, $\langle s_n, \boldsymbol{V}^n\rangle \in P$.
- p is *P-eventual* if there exists n such that $\langle s_n, \boldsymbol{V}^n\rangle \in P$.
- p is *P-almost-always* if there exists n such that for all $n' > n$, $\langle s_{n'}, \boldsymbol{V}^{n'}\rangle \in P$.

The $\exists$-Presburger-i.o. (resp. always, eventual and almost-always) problem for NCMF M is to decide whether the following statement holds:

there is an ω-path p starting from I that is P-i.o. (resp. P-always, P-eventual and P-almost-always).

The $\forall$-Presburger-i.o. (resp. always, eventual and almost-always) problem for NCMF M is to decide whether the following statement holds:

for every ω-path p, if p starts from I, then p is P-i.o. (resp. P-always, P-eventual and P-almost-always).

We use $\boldsymbol{X}$ to denote the vector of the $k+1$ counters $x_0, x_1, \cdots, x_k$ in M, with x_0 the free counter and with $x_1, \cdots, x_k$ the reversal-bounded counters.

3 Decidable Results

In this section, we show that both the $\exists$-Presburger-i.o. problem and the $\exists$-Presburger-eventual problem are decidable for NCMFs.

3.1 The $\exists$-Presburger-i.o. Problem Is Decidable

The $\exists$-Presburger-i.o. problem is to determine the existence of an ω-path p (called a *witness*) $\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle \cdots$ of an NCMF M such that p is P-i.o. with respect to I. Since P is a Presburger definable set of configurations, by definition, $P(\boldsymbol{X}, s)$ can be written in a disjunctive normal form, $\bigvee P_i(\boldsymbol{X}, s)$, where each $P_i(\boldsymbol{X}, s)$ is a linear-mod-conjunction of atomic linear constraints and atomic mod-constraints over counters and control states (control states are encoded as bounded integers) in M. Obviously, p is P-i.o. iff p is P_i-i.o. for some i. Therefore, without loss of generality, we assume P itself is a linear-mod-conjunction.

There are only finitely many control states $S = \{\hat{s}_1, \cdots, \hat{s}_m\}$ in M. Therefore, $P(\boldsymbol{X}, s)$ can be written as $\bigvee_{\hat{s}_i \in S} s = \hat{s}_i \wedge P(\boldsymbol{X}, \hat{s}_i)$. p is P-i.o. iff p is $P(\cdot, \hat{s}_i)$-i.o. *on some control state $\hat{s}_i$*: there are infinitely many n such that $s_n = \hat{s}_i$ and $P(\boldsymbol{V}^n, \hat{s}_i)$. Therefore, the $\exists$-Presburger-i.o. problem is reduced to the problem of deciding whether there exist a control state s and a witness p starting from I such that $p = \langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle \cdots$ is P-i.o. on s, where P is a linear-mod-conjunction on counters $\boldsymbol{X}$ only. Assume that $P(\boldsymbol{X})$ is $P^{\text{linear}}(\boldsymbol{X}) \wedge P^{\text{mod}}(\boldsymbol{X})$, where P^{linear} is a linear-conjunction over $\boldsymbol{X}$ and P^{mod} is a mod-conjunction over $\boldsymbol{X}$. The following lemma states that, as far as an infinite often property is concerned, P^{mod} can be eliminated by building "mod" into the control states of M.

Lemma 1. *Given M (an NCMF with counters $\boldsymbol{X}$ and with control states S), I (a Presburger-definable set of configurations of M), P (a linear-mod-conjunction over $\boldsymbol{X}$), and s (a control state in S), we can effectively construct M' (an NCMF with*

counters $\boldsymbol{X}$ and with control states S'), I' (a Presburger-definable set of configurations of M'), P' (a linear-conjunction over $\boldsymbol{X}$), and s' (a control state in S'), such that the following two statements are equivalent:

- *In M, there exists a witness p starting from I such that p is P-i.o. on state s,*
- *In M', there exists a witness p' starting from I' such that p' is P'-i.o. on state s'.*

Because of Lemma 1, it suffices to investigate the existence of a P-i.o witness p on state s with P in the form of a linear-conjunction over m linear constraints:

$$\sum_{0 \le i \le k} a_{ij} x_i \#_j b_j \tag{1}$$

where $\#_j$ stands for $>$ or $=$, for $1 \le j \le m$. We use $\mathbf{A}$, $\#$, and $\mathbf{b}$ to denote the coefficient matrix (m by $k+1$) of a_{ij}, the column (m by 1) of comparisons $\#_j$, and the column (m by 1) of numbers b_j. Thus, P shown in (1) can be written as

$$\mathbf{A}\boldsymbol{X}\#\mathbf{b}. \tag{2}$$

We say $(k+1)$-ary vector $\boldsymbol{\Delta}$ is *P-positive* if $\mathbf{A}\boldsymbol{\Delta} \ge \mathbf{0}$. From definition, an ω-path p of $\langle s_0, \boldsymbol{V}^0\rangle \cdots \langle s_n, \boldsymbol{V}^n\rangle \cdots$ is a desired witness iff the following conditions are satisfied:

(IO-1). p starts from I; i.e., $I(s_0, \boldsymbol{V}^0)$ holds,
(IO-2). There are infinitely many numbers $n_1, \cdots, n_i, \cdots$ (with $0 < n_1 < \cdots < n_i < \cdots$) such that $s_{n_i} = s$ and $P(\boldsymbol{V}^{n_i})$ for each i.

The following lemma states that condition (IO-2) can be strengthened: for each i, $\boldsymbol{V}^{n_{i+1}} - \boldsymbol{V}^{n_i}$ is P-positive.

Lemma 2. *Let P be a linear conjunction as in (2). Let s be a state in an NCMF M. For any ω-path p of M, condition (IO-2) is equivalent to the following condition:*

(IO-2′). There are infinitely many numbers $n_1, \cdots, n_i, \cdots$ (with $0 < n_1 < \cdots < n_i < \cdots$) such that $s_{n_i} = s$, $P(\boldsymbol{V}^{n_i})$, and $\boldsymbol{V}^{n_{i+1}} - \boldsymbol{V}^{n_i}$ is P-positive, for each i.

Up to now, we have not used the condition that counters $x_1, \cdots, x_k$ are reversal-bounded and that counter x_0 is free. Let C be the largest absolute value of the integer constants appearing in all the tests in M. The idea is that, on the ω-path p, each reversal-bounded counter will eventually behave as a 0-reversal-bounded (i.e., either nondecreasing or nonincreasing) counter after the last reversal has been made. Once a reversal-bounded counter behaves as 0-reversal-bounded, it will either stay unchanged between $-C$ and C forever, or move beyond C (or $-C$) and never come back. That is, there is n_0 such that each reversal-bounded counter x_i, $1 \le i \le k$, has one of the following $2C+3$ *modes*:

(MD1-c) with $-C \le c \le C$. For all $n \ge n_0$, $\boldsymbol{V}_i^n = \boldsymbol{V}_i^{n+1} = c$. That is, x_i is always c that is between $-C$ and C after n_0,
(MD2). For all $n \ge n_0$, $C < \boldsymbol{V}_i^n \le \boldsymbol{V}_i^{n+1}$. That is, x_i is nondecreasing and always greater than C,
(MD3). For all $n \ge n_0$, $-C > \boldsymbol{V}_i^n \ge \boldsymbol{V}_i^{n+1}$. That is, x_i is nonincreasing and always smaller than $-C$.

Let *mode vector* $\boldsymbol{\theta} \in (\{\text{MD1-}c : -C \leq c \leq C\} \cup \{\text{MD2}, \text{MD3}\})^k$ assign to each reversal-bounded counter x_i a mode $\boldsymbol{\theta}_i$. Each ω-path p has a unique mode vector. Now we fix any mode vector $\boldsymbol{\theta}$.

M can be effectively modified into an NCMF $M^{\boldsymbol{\theta}}$ such that the reversal-bounded counters in $M^{\boldsymbol{\theta}}$ behave according to the mode vector $\boldsymbol{\theta}$. An edge $\langle s^1, t, \mathbf{incr}, s^2\rangle$ in M is *compatible* with a mode vector $\boldsymbol{\theta}$, if, for each reversal-bounded counter x_i with $1 \leq i \leq k$, the following conditions hold:

- If x_i is in mode MD1-c for some $-C \leq c \leq C$, x_i will not change on the edge; i.e., $\mathbf{incr}_i = 0$ if $\boldsymbol{\theta}_i =$MD1-c,
- If x_i is in mode MD2, x_i will not decrease on the edge; i.e., $\mathbf{incr}_i \geq 0$ if $\boldsymbol{\theta}_i =$ MD2,
- If x_i is in mode MD3, x_i will not increase on the edge; i.e., $\mathbf{incr}_i \leq 0$ if $\boldsymbol{\theta}_i =$ MD3,

The modification starts with deleting all the edges in M that are not compatible with $\boldsymbol{\theta}$ from M. Then, more tests are added to the remaining edges to make sure that the reversal-bounded counters always have the desired values. More precisely, for each x_i with $1 \leq i \leq k$ and for each remaining edge $\langle s^1, t, \mathbf{incr}, s^2\rangle$ in M, if x_i is in mode MD2, then we add a test of $x_i > C$ to the original test t of the edge. Doing this will guarantee that the values of x_i before and after this edge are greater than C (no matter whether $\mathbf{incr}_i = 0$ or $\mathbf{incr}_i = 1$). The cases when x_i is in mode MD3 can be handled similarly. If, however, x_i is in mode MD1-c for some $-C \leq c \leq C$, we simply add a test of $x_i = c$ to the original test t of the edge. The result $M^{\boldsymbol{\theta}}$ is also an NCMF with 0-reversal-bounded counters.

Obviously, from the choice of constant C, $M^{\boldsymbol{\theta}}$ is insensitive to the actual starting values of the 0-reversal-bounded counters. That is, if (1) $\langle s^1, \boldsymbol{V}^1\rangle$ can reach $\langle s^2, \boldsymbol{V}^1 + \boldsymbol{\Delta}^1\rangle$ through a path p_1 in $M^{\boldsymbol{\theta}}$, and (2) $\langle s^2, \boldsymbol{V}^2\rangle$ can reach $\langle s^3, \boldsymbol{V}^2 + \boldsymbol{\Delta}^2\rangle$ through a path p_2 in $M^{\boldsymbol{\theta}}$, such that the free counter x_0 has the same value at the end of p_1 and at the beginning of p_2, i.e., $\boldsymbol{V}_0^1 + \boldsymbol{\Delta}_0^1 = \boldsymbol{V}_0^2$, then each 0-reversal-bounded counter x_i with $1 \leq i \leq k$ in p_2 can start from $\boldsymbol{V}_i^1 + \boldsymbol{\Delta}_i^1$ (instead of $\boldsymbol{V}_i^2$) and at the end of p_2, x_i has value $\boldsymbol{V}_i^1 + \boldsymbol{\Delta}_i^1 + \boldsymbol{\Delta}_i^2$ (instead of $\boldsymbol{V}_i^2 + \boldsymbol{\Delta}_i^2$). Thus, path p_1 can be extended according to path p_2. The reason is that after changing the starting value of x_i, the test of x_i on each edge on path p_2 gives the same truth value as the old starting value, and, hence, path p_2 can be perfectly followed after p_1. This is summarized in the following technical lemma.

Lemma 3. *For any control states s^1, s^2 and s^3, for any mode vector $\boldsymbol{\theta}$, for any (k+1)-ary vectors $\boldsymbol{V}^1$, $\boldsymbol{V}^2$, $\boldsymbol{\Delta}^1$, and $\boldsymbol{\Delta}^2$, if $\boldsymbol{V}_0^1 + \boldsymbol{\Delta}_0^1 = \boldsymbol{V}_0^2$, $\langle s^1, \boldsymbol{V}^1\rangle \rightsquigarrow_{M^{\boldsymbol{\theta}}} \langle s^2, \boldsymbol{V}^1 + \boldsymbol{\Delta}^1\rangle$ and $\langle s^2, \boldsymbol{V}^2\rangle \rightsquigarrow_{M^{\boldsymbol{\theta}}} \langle s^3, \boldsymbol{V}^2 + \boldsymbol{\Delta}^2\rangle$, then $\langle s^2, \boldsymbol{V}^1 + \boldsymbol{\Delta}^1\rangle \rightsquigarrow_{M^{\boldsymbol{\theta}}} \langle s^3, \boldsymbol{V}^1 + \boldsymbol{\Delta}^1 + \boldsymbol{\Delta}^2\rangle$.*

Let $I' = \{\beta : \exists \alpha \in I(\alpha \rightsquigarrow_M \beta)\}$. I' is the set of reachable configurations from configurations in I. From Theorem 1, I' is Presburger. The following lemma states that the $\exists$-Presburger i.o. problem of M can be reduced to one for 0-reversal-bounded NCMFs $M^{\boldsymbol{\theta}}$.

Lemma 4. *There exists a witness p in M starting from I that is P-i.o. at state s iff for some mode vector $\boldsymbol{\theta}$, there exists a witness p' in $M^{\boldsymbol{\theta}}$ starting from I' that is P-i.o. at state s.*

For any state s and mode vector $\boldsymbol{\theta}$, we define a predicate $Q^{s,\boldsymbol{\theta}}(v, v')$ as follows. $Q^{s,\boldsymbol{\theta}}(v, v')$ iff there exist two vectors $\boldsymbol{V}$ and $\boldsymbol{\Delta}$ such that the following statements are satisfied: (Q1). v and v' are the values of the free counter; i.e., $v = \boldsymbol{V}_0$ and $v' = \boldsymbol{V}_0 + \boldsymbol{\Delta}_0$; (Q2). Both $\boldsymbol{V}$ and $\boldsymbol{V} + \boldsymbol{\Delta}$ satisfy P; i.e., $P(\boldsymbol{V}) \wedge P(\boldsymbol{V} + \boldsymbol{\Delta})$; (Q3). Configuration $\langle s, \boldsymbol{V}\rangle$ can reach configuration $\langle s, \boldsymbol{V} + \boldsymbol{\Delta}\rangle$ in $M^{\boldsymbol{\theta}}$; i.e., $\langle s, \boldsymbol{V}\rangle \rightsquigarrow_{M^{\boldsymbol{\theta}}} \langle s, \boldsymbol{V} + \boldsymbol{\Delta}\rangle$; (Q4). $\boldsymbol{\Delta}$ is P-positive; (Q5). Finally, configuration $\langle s, \boldsymbol{V}\rangle$ is reachable from some configuration in I; i.e., $\langle s, \boldsymbol{V}\rangle \in I'$.

Lemma 5. *For any state s and mode vector $\boldsymbol{\theta}$, $Q^{s,\boldsymbol{\theta}}$ is Presburger.*

It is easy to check that $Q^{s,\boldsymbol{\theta}}$ is transitive.

Lemma 6. *For any state s and mode vector $\boldsymbol{\theta}$, $Q^{s,\boldsymbol{\theta}}$ is transitive. That is, for all integers v_1, v_2, and v_3, $Q^{s,\boldsymbol{\theta}}(v_1, v_2) \wedge Q^{s,\boldsymbol{\theta}}(v_2, v_3)$ implies $Q^{s,\boldsymbol{\theta}}(v_1, v_3)$.*

Before we go any further, we need to uncover the intuitive meaning underlying the definition of $Q^{s,\boldsymbol{\theta}}$. $Q^{s,\boldsymbol{\theta}}(v, v')$ indicates the following scenario. Through a path in $M^{\boldsymbol{\theta}}$, $M^{\boldsymbol{\theta}}$ can send the free counter x_0 from value v to v', with some properly chosen starting values for the 0-reversal-bounded counters ((Q1) and (Q3)). On the path, $M^{\boldsymbol{\theta}}$ starts from control state s and finally moves back to the same control state, as given in (Q3). Therefore, this path is a loop on the control state s. It is noticed that the starting configuration and the ending configuration of the path both satisfy P (as given in (Q2)), and in particular, the counter changes $\boldsymbol{\Delta}$ is P-positive (as given in (Q4)).

If we can repeat the loop, then the resulting ω-path is P-i.o. (this is because of Lemma 2 and the fact that $\boldsymbol{\Delta}$ is P-positive.) and, from (Q5), starts from I'. However, this loop may not repeat. The reason is that the starting value v of the free counter decides the path of the loop and therefore, when $M^{\boldsymbol{\theta}}$ executes the loop for a second time, the starting value v' of the free counter may lead to a different path. Thus, trying to repeat the same loop is too naive. However, the key technique shown below attempts to concatenate infinitely many (different) loops into an ω-path that is a P-i.o. witness.

Let v^ω be an ω-sequence of integers

$$v_0, v_1, \cdots, v_n, \cdots.$$

v^ω is an *ω-chain* of $Q^{s,\boldsymbol{\theta}}$ if $Q^{s,\boldsymbol{\theta}}(v_n, v_{n+1})$ holds for all $n \geq 0$. According to Lemma 6, $Q^{s,\boldsymbol{\theta}}$ is transitive. Therefore, if v^ω is an ω-chain then $Q^{s,\boldsymbol{\theta}}(v_n, v_m)$ holds for any $n < m$. The following lemma states that the existence of an ω-chain for $Q^{s,\boldsymbol{\theta}}$ is decidable.

Lemma 7. *It is decidable whether a transitive Presburger predicate over two variables has an ω-chain. Thus, from Lemma 5 and Lemma 6, it is decidable whether $Q^{s,\boldsymbol{\theta}}$ has an ω-chain.*

We now show that the existence of a P-i.o witness p at state s starting from I is equivalent to the existence of an ω-chain of $Q^{s,\boldsymbol{\theta}}$ for some mode vector $\boldsymbol{\theta}$.

Lemma 8. *There is an ω-path p that is P-i.o. at state s and starts from I iff, for some mode vector $\boldsymbol{\theta}$, $Q^{s,\boldsymbol{\theta}}$ has an ω-chain.*

Finally, combining Lemma 7 and Lemma 8, we have,

Theorem 2. *The $\exists$-Presburger-i.o. problem is decidable for reversal-bounded multicounter machines with a free counter.*

The $\exists$-Presburger-i.o. problem is equivalent to the negation of the $\forall$-Presburger-almost-always problem. Thus,

Theorem 3. *The $\forall$-Presburger-almost-always problem is decidable for reversal bounded multicounter machines with a free counter.*

3.2 The $\exists$-Presburger-Eventual Problem Is Decidable

Given two Presburger-definable sets I and P of configurations for NCMF M, the $\exists$-Presburger-eventual problem is to decide whether there exists a P-eventual ω-path p starting from I. Recall that the Presburger-definable set I' is the set of all configurations in P that are reachable from a configuration in I. In the following lemma, **true** means the set of all configurations. It is easy to see that

Lemma 9. *There is a P-eventual ω-path starting from I iff there is a* **true***-i.o. ω-path starting from I'.*

Hence, combining Lemma 9 and Theorem 2, we have,

Theorem 4. *The $\exists$-Presburger-eventual problem is decidable for reversal-bounded multicounter machines with a free counter.*

Since the $\exists$-Presburger-eventual problem is equivalent to the negation of the $\forall$-Presburger-always problem, we have,

Theorem 5. *The $\forall$-Presburger-always problem is decidable for reversal-bounded multicounter machines with a free counter.*

The Presburger safety analysis problem is slightly different from the $\forall$-Presburger-always problem: the former looks at (finite) paths, while the latter looks at ω-paths.

4 Undecidability Results

In this section, we point out that both the $\exists$-Presburger-always problem and the $\exists$-Presburger-almost-always problem are undecidable for 0-reversal-bounded NCMs. Obviously, the undecidability remains when NCMFs are considered.

In [12], it is shown that the $\exists$-Presburger-always problem and the $\exists$-Presburger-almost-always problem are undecidable for discrete timed automata. The following techniques are used in that paper:

- A deterministic two-counter machine can be simulated by a generalized discrete timed automaton that allows tests in the form of linear constraints,
- The generalized discrete timed automaton can be simulated by a discrete timed automaton under a Presburger path restriction P [12] (i.e., each intermediate configuration of the discrete timed automaton must be in P),

- The halting problem (i.e., whether a control state is reachable, which is undecidable) for deterministic two-counter machines can be reduced to the $\exists$-Presburger-always problem for discrete timed automata,
- The finiteness problem (which is undecidable) for deterministic two-counter machines can be reduced to the $\exists$-Presburger-almost-always problem for discrete timed automata.

If in the items above, "discrete timed automaton" is replaced by "0-reversal-bounded multicounter machine", the techniques are still applicable. The reason is that, as shown below, any deterministic two-counter machine can be simulated by a deterministic generalized 0-reversal-bounded multicounter machine that allows tests in the form of linear constraints on counters.

Lemma 10. *Any deterministic two-counter machine M can be simulated by a deterministic 0-reversal-bounded multicounter machine M' that allows tests in the form of $y - z \# c$, where y and z are counters, and c is an integer [18].*

Analogous to the proofs in [12], we have,

Theorem 6. *The $\exists$-Presburger-always problem and the $\exists$-Presburger-almost-always problem are undecidable for 0-reversal-bounded multicounter machines. The undecidability remains when reversal-bounded multicounter machines with a free counter are considered.*

Considering the negations of the two problems, we have,

Theorem 7. *The $\forall$-Presburger-eventual problem and the $\forall$-Presburger-i.o. problem are undecidable for 0-reversal-bounded multicounter machines. The undecidability remains when reversal-bounded multicounter machines with a free counter are considered.*

5 Conclusions

In this paper, we investigated a number of Presburger liveness problems for NCMFs. We showed that

- The $\exists$-Presburger-i.o. problem and the $\exists$-Presburger-eventual problem are both decidable. So are their duals, the $\forall$-Presburger-almost-always problem and the $\forall$-Presburger-always problem.
- The $\forall$-Presburger-i.o. problem and the $\forall$-Presburger-eventual problem are both undecidable. So are their duals, the $\exists$-Presburger-almost-always problem and the $\exists$-Presburger-always problem.

These results can be used to formulate a weak form of Presburger linear temporal logic and develop its model-checking theories for NCMFs. We believe the techniques developed in [12] and in this paper can be naturally combined to study the same set of liveness problems on an extended form of discrete timed automata containing, besides clocks, a number of reversal-bounded counters and a free counter. We conjecture that the $\exists$-Presburger-i.o. problem is also decidable for (discrete timed) pushdown automata

when the counts on individual stack symbols are part of the Presburger property being verified [8].

As for applications of NCMFs, "reversal-bounded counters" may appear unnatural, and applying the decidable results presented in this paper in model-checking may seem remote. However, the model of NCMFs does have applications in verification/debugging infinite state systems as we discuss below.

- Many infinite state systems can be modeled as multicounter machines. These machines, usually having Turing computing power, can be approximated by NCMFs by restricting all but one counter to be reversal-bounded. This approximation technique provides a way to debug Presburger safety properties for, for instance, arithmetic programs (for a number of conservative approximation techniques for real-time systems see [9,10,11]). On the other hand, the technique also shows a way to verify an $\exists$-Presburger i.o. problem for a multicounter machine if the same problem is true on the resulting NCMF.
- A non-decreasing counter is also a reversal-bounded counter with zero reversal bound. This kind of counters has a lot of applications. For instance, it can be used to count time elapse, the number of external events, the number of a particular branch taken by a nondeterministic program (this is important, when fairness is taken into account), etc. For example, consider a finite-state transition system T. Associate a name 'a' from a finite alphabet to each transition in T (a, in the *reactive* system T, can be treated as the input signal triggering the transition). At any moment in an execution of T, $\#_a$ is used to count the number of transitions labeled by a that have been executed. Each $\#_a$ can be considered as a 0-reversal-bounded counter, since $\#_a$ is nondecreasing along any execution path. To make the system more complex, on some transitions, the triggering conditions also contain a test that compares $\#_b - \#_c$ against an integer constant, for some fixed labels b and c [1]. Essentially T can be treated as a NCMF: those counts of $\#_a$'s are reversal-bounded counters and $\#_b - \#_c$ is the free counter. The results in this paper show that the following statement can be automatically verified:

  There is an execution of T such that $\#_a + 2\#_b - 5\#_c > 0$ holds for infinitely many times.

  This result can be used to argue whether a fairness condition on the event label counts of T is realistic.

The decision procedure for the $\exists$-Presburger i.o. problem seems hard to implement. However, by closely looking at the proofs, the hard part is how to (practically) calculate the binary reachability of an NCMF. Once this is done, testing the existence of the ω-chain in Lemma 7 and Lemma 8 is equivalent to checking a Presburger predicate (i.e., $Q^{s,\boldsymbol{\theta}}$ in the lemmas) in a particular format (the Omega Library [22] can be used to do the checking). Calculating the binary reachability of an NCMF needs some software engineering thoughts. We are currently conducting a prototype tool implementation.

Thanks go to anonymous reviewers for many useful suggestions.

[1] It is important that b and c are fixed. If we allow comparisons on the counts of four labels (i.e., besides the test on $\#_b - \#_c$, we have a test on $\#_d - \#_e$), then T is Turing powerful [18].

References

1. R. Alur and D. Dill, "The theory of timed automata," *TCS*, 126(2):183-236, 1994
2. P. Abdulla and B. Jonsson, "Verifying programs with unreliable channels," *Information and Computation*, 127(2): 91-101, 1996
3. A. Bouajjani, J. Esparza, and O. Maler. "Reachability analysis of pushdown automata: application to model-Checking," *CONCUR'97*, **LNCS** 1243, pp. 135-150
4. G. Cece and A. Finkel, "Programs with Quasi-Stable Channels are Effectively Recognizable," *CAV'97*, **LNCS** 1254, pp. 304-315
5. H. Comon and Y. Jurski. "Multiple counters automata, safety analysis and Presburger arithmetic," *CAV'98*, **LNCS** 1427, pp. 268-279
6. H. Comon and Y. Jurski, "Timed automata and the theory of real numbers," *CONCUR'99*, **LNCS** 1664, pp. 242-257
7. Z. Dang, "Binary reachability analysis of timed pushdown automata with dense clocks," *CAV'01*, **LNCS** 2102, pp. 506-517
8. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer and J. Su, "Binary Reachability Analysis of Discrete Pushdown Timed Automata," *CAV'00*, **LNCS** 1855, pp. 69-84
9. Z. Dang, O. H. Ibarra and R. A. Kemmerer, "Decidable Approximations on Generalized and Parameterized Discrete Timed Automata," *COCOON'01*, **LNCS** 2108, pp. 529-539
10. Z. Dang and R. A. Kemmerer, "Using the ASTRAL model checker to analyze Mobile IP," *Proceedings of the Twenty-first International Conference on Software Engineering (ICSE'99)*, pp. 132-141, IEEE Press, 1999
11. Z. Dang and R. A. Kemmerer, "Three approximation techniques for ASTRAL symbolic model checking of infinite state real-time systems," *Proceedings of the Twenty-second International Conference on Software Engineering (ICSE'00)*, pp. 345-354, IEEE Press, 2000
12. Z. Dang, P. San Pietro, and R. A. Kemmerer, "On Presburger Liveness of Discrete Timed Automata," *STACS'01*, **LNCS** 2010, pp. 132-143
13. A. Finkel and G. Sutre. "Decidability of Reachability Problems for Classes of Two Counter Automata," *STACS'00*, **LNCS** 1770, pp. 346-357
14. A. Finkel, B. Willems, and P. Wolper. "A direct symbolic approach to model checking pushdown systems," *INFINITY'97*
15. O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, **25** (1978) 116-133
16. O. H. Ibarra, "Reachability and safety in queue systems with counters and pushdown stack," *Proceedings of the International Conference on Implementation and Application of Automata*, pp. 120-129, 2000
17. O. H. Ibarra, Z. Dang, and P. San Pietro, "Verification in Loosely Synchronous Queue-Connected Discrete Timed Automata," submitted. 2001
18. O. H. Ibarra, J. Su, T. Bultan, Z. Dang, and R. A. Kemmerer, "Counter Machines: Decidable Properties and Applications to Verification Problems,", *MFCS'00*, **LNCS 1893**, pp. 426-435
19. G. Kreisel and J. L. Krevine, "Elements of Mathematical Logic," North-Holland, 1967
20. K. L. McMillan. "Symbolic model-checking - an approach to the state explosion problem," PhD thesis, Department of Computer Science, Carnegie Mellon University, 1992
21. W. Peng and S. Purushothaman. "Analysis of a Class of Communicating Finite State Machines," *Acta Informatica*, 29(6/7): 499-522, 1992
22. W. Pugh, "The Omega test: a fast and practical integer programming algorithm for dependence analysis," *Communications of the ACM*, 35(8): 102-114, 1992

A Mechanically Verified Compiling Specification for a Lisp Compiler*

Axel Dold and Vincent Vialard

Fakultät für Informatik
Universität Ulm
D-89069 Ulm, Germany
Fax: +49/(0)731/50-24119
{dold|vialard}@informatik.uni-ulm.de

Abstract. We report on an ongoing effort in mechanically proving correct a compiling specification for a bootstrap compiler from ComLisp (a subset of ANSI Common Lisp sufficiently expressive to serve as a compiler implementation language) to binary Transputer code using the PVS system. The compilation is carried out in four steps through a series of intermediate languages. This paper focuses on the first phase, namely, the compilation of ComLisp to the stack-intermediate language SIL, where parameter passing is implemented by a stack technique. The context of this work is the joint research effort *Verifix* aiming at developing methods for the construction of correct compilers for realistic programming languages.

1 Introduction

The use of computer based systems for safety-critical applications requires high dependability of the software components. In particular, it justifies and demands the verification of programs typically written in high-level programming languages. Correct program execution, however, crucially depends on the correctness of the binary machine code executable, and therefore, on the correctness of system software, especially compilers. As already noted in 1986 by Chirica and Martin [3], full compiler correctness comprises both the correctness of the compiling specification (with respect to the semantics of the languages involved) as well as the correct implementation of the specification.

Verifix [9, 6] is a joint German research effort of groups at the universities Karlsruhe, Kiel, and Ulm. The project aims at developing innovative methods for constructing provably correct compilers which generate efficient code for realistic, practically relevant programming languages. These realistic compilers are to be constructed using approved development techniques. In particular, even standard unverified compiler generation tools (such as Lex or Yacc) may be used, the correctness of the generated code being verified at compile time using verified

* This research has been funded by the Deutsche Forschungsgemeinschaft (DFG) under project "*Verifix*".

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 144–155, 2001.
© Springer-Verlag Berlin Heidelberg 2001

program checkers [7]. *Verifix* assumes hardware to behave correctly as described in the instruction manuals.

In order not to have to write the verified parts of the compiler and checkers directly in machine code, a fully verified and correctly implemented *initial compiler* is required, for which efficiency of the produced code is not a priority. The initial correct compiler to be constructed in this project transforms ComLisp programs into binary Transputer code. ComLisp is an imperative proper subset of ANSI-Common Lisp and serves both as a source and implementation language for the compiler. The construction process of the initial compiler consists of the following steps:

- define syntax and semantics of appropriate intermediate languages.
- define the compiling specification, a relation between source and target language programs and prove (with respect to the language semantics) its correctness according to a suitable correctness criterion.
- construct a correct compiler implementation in the source language itself (a transformational constructive approach is applied which builds a correct implementation from the specification by stepwise applying correctness-preserving development steps [5]).
- use an existing (unverified) implementation of the source language (here: some arbitrary Common Lisp compiler) to execute the program. Apply the program to itself and bootstrap a compiler executable. Check syntactically, that the executable code has been generated according to the compiling specification. For this last step, a realistic technique for low level compiler verification has been developed which is based on rigorous a posteriori syntactic code inspection [8,11]. This closes the gap between high-level implementation and executable code.

The size and complexity of the verification task in constructing a correct compiler is immense. In order to manage it, suitable mechanized support for both specification and verification is necessary. We have chosen the PVS specification and verification system [16] to support the verification of the compiling specification and the construction process of a compiler implementation in the source language.

In this paper, we focus on the mechanical verification of the compiling specification for the ComLisp compiler. In particular, we describe the formalization and verification process of the first compilation phase from ComLisp to the stack-based intermediate language SIL, the first of a series of intermediate languages used to compile ComLisp programs into binary Transputer machine code:

$$\text{ComLisp} \rightarrow \text{SIL} \rightarrow \text{C}^{\text{int}} \rightarrow \text{TASM} \rightarrow \text{TC}$$

First, ComLisp is translated into a stack intermediate language (SIL), where parameter passing is implemented by a stack technique. Expressions are transformed from a prefix notation into a postfix notation according to the stack principle. SIL is then compiled into C^{int} where the ComLisp data structures (s-expressions) and operators are implemented in linear integer memory using a

run-time stack and a heap. These two steps are machine independent. In the next step, control structures of C^{int} are implemented by linear assembler code with jumps, and finally, abstract assembler code is transformed into binary Transputer code.

This paper is organized as follows. The next section presents the formalization of the languages ComLisp and SIL, that is, their abstract syntax and semantics. Section 3 then focuses on the compilation process from ComLisp to SIL. Finally, Section 4 is concerned with the correctness of this compilation process.

2 Syntax and Semantics of the Languages

2.1 ComLisp

A ComLisp program consists of a list of global variables, a list of possibly mutual recursive function definitions, and a main form. ComLisp forms (expressions) include the *abort* form, s-expression constants, variables, assignments, sequential composition (*progn*), conditional, while loop, call of user defined functions, call of built-in unary (*uop*) and binary (*bop*) ComLisp operators, local let-blocks, *list*$*$ operator (constructing a s-expression list from its evaluated arguments), case-instruction, and instructions for reading from the input sequence and writing to the output. The ComLisp operators include the standard operators for lists (e.g. *length*), type predicates for the different kinds of s-expressions, and the standard arithmetic operations (e.g. $+, *, \mathit{floor}$). The only available datatype is the type of s-expressions which are binary trees built with constructor "cons", where the leaves are either integers, characters, strings, or symbols. The set of symbols includes T and *NIL*. The abstract syntax of ComLisp is given as follows:

$$
\begin{array}{lcl}
p & ::= & x_1, \ldots, x_k; f_1, \ldots, f_n; e \\
f & ::= & h(x_1, \ldots, x_m) \leftarrow e \\
e & ::= & \mathit{abort} \mid c \mid x \mid x := e \mid \mathit{progn}(e_1, \ldots, e_n) \mid \mathit{if}(e_1, e_2, e_3) \mid \mathit{while}(e_1, e_2) \mid \\
 & & \mathit{call}(h, e_1, \ldots, e_n) \mid \mathit{uop}(e) \mid \mathit{bop}(e_1, e_2) \mid \mathit{let}(x_1 = e_1, \ldots, x_n = e_n; e) \mid \\
 & & \mathit{list}*(e_1, \ldots, e_n) \mid \mathit{cond}(p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n) \mid \\
 & & \mathit{read_char} \mid \mathit{peek_char} \mid \mathit{print_char}(e)
\end{array}
$$

The static semantics of ComLisp programs, function definitions, and forms is specified by means of several well-formedness predicates. A ComLisp form is *well-formed*—with respect to a local variable environment ζ (a list of formal parameters), a list of global variables γ, and a function environment Γ (a list of function definitions)—if the list of local and global variables are disjoint, all variables are declared (that is, occur either in ζ or γ) and each user-defined function is declared in Γ and called with the correct number of arguments (correct parameter passing). Formally, a relation $\mathit{wf}(e, \zeta, \gamma, \Gamma)$ is defined inductively on the structure of forms (omitted here). Analogously, well-formedness relations for function environments (predicate $\mathit{wf}_{\mathrm{proc}}(\Gamma, \gamma)$) and programs (predicate $\mathit{wf}_{\mathrm{program}}(p)$) are defined (definitions omitted).

For the intermediate languages occurring in the different compilation phases of the ComLisp to Transputer compiler, a uniform relational semantics description has been chosen. The (dynamic) semantics of ComLisp is defined in a structural operational way by a set of inductive rules for the different ComLisp forms. This kind of semantics is also referred to as *big-step semantics* or *evaluation semantics* in contrast to a transition semantics (small-step semantics) such as abstract state machines (ASM's). A ComLisp *state* is a triple consisting of an (infinite) input sequence (stream) of characters, an output list of characters, and the variable state which is a mapping from identifiers to values (s-expressions):

$$state_{\mathrm{CL}} ::= sequence[char] \times char^* \times (Ident \to SExpr)$$

ComLisp forms are expressions with side-effects, that is, they denote state transformers transforming states to pairs of result value and result state. The definition of the semantics of forms uses the following notation: $\Gamma \vdash s : e \to (v, q)$. It states that evaluating form e in state s and function environment Γ terminates and results in a value v and final state q. Given rules for each kind of form, the semantics is defined as the smallest relation $\to$ satisfying the set of rules. For example, the semantics of a function call is given by two rules. One for parameterless functions (omitted here), and one for functions with parameters, where the parameters are sequentially evaluated, the resulting values being then bound to the parameters before evaluation of the body and unbound after returning the value:

$$\frac{\begin{array}{c}[f(x_1 \cdots x_n) \leftarrow body] \in \Gamma \ (n \geq 1) \\ \Gamma \vdash q_i : e_i \to (v_i, q_{i+1}) \ (1 \leq i \leq n) \\ \Gamma \vdash q_{n+1}[x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n] : body \to (v, r)\end{array}}{\Gamma \vdash q_1 : call(f, e_1, \ldots, e_n) \to (v, r[x_1 \leftarrow q_{n+1}(x_1), \ldots, x_n \leftarrow q_{n+1}(x_n)])}$$

The semantics of a ComLisp program is given by the input/output behavior of the program defined by a relation $P_{\mathrm{sem_{CL}}}$ between input streams is and output lists ol. $P_{\mathrm{sem_{CL}}}(p)(is, ol)$ holds if the evaluation of the main form e in an initial state, where the input stream is given by is, the output list is empty and all variables are initialized with NIL, terminates with a value v in some state q with output list ol. Formally:

$$P_{\mathrm{sem_{CL}}}(p)(is, ol) ::= \exists v, q.\ (\Gamma \vdash (is, [], \lambda x.NIL) : e \to (v, q)) \ \wedge \ (q_{\mathrm{output}} = ol)$$

2.2 SIL

SIL, the stack intermediate language, is a language with parameterless procedures and s-expressions as available datatype. Programs operate on a runtime stack with frame-pointer relative addresses. A SIL program consists of a list of parameterless procedure declarations and a main statement. There are no variables, only memory locations and the machine has statements for copying values from the global to the local memory and vice versa. For example, $copy(i, j)$ copies the content at stack relative position i to relative position j, $gcopy(g, i)$ copies

from the global memory at position g to the relative position i, and $itef(i, s_1, s_2)$ executes instruction s_2 if the content of stack relative position i is NIL, otherwise s_1 is executed.

$$
\begin{array}{lll}
p & ::= & f_1, \ldots, f_n; s \\
f & ::= & h \leftarrow s \\
s & ::= & abort \mid copyc(c, i) \mid copy(i, j) \mid gcopy(g, i) \mid copyg(g, i) \mid \\
 & & itef(i, s_1, s_2) \mid sq(s_1, \ldots, s_n) \mid fcall(h, i) \mid uop(i) \mid bop(i) \mid \\
 & & while(i, s_1, s_2) \mid read_char(i) \mid peek_char(i) \mid print_char(i) \mid list*(n, i)
\end{array}
$$

The static semantics is again specified by means of well-formedness predicates for SIL statements, SIL procedure declarations, and SIL programs (definitions omitted here). SIL statements denote state transformers, where a SIL state consists of the input stream, the output list, the global memory (a list of s-expressions), and the local memory (consisting of the frame pointer $base : Nat$ and the stack, a function from natural numbers to s-expressions).

$$state_{\text{SIL}} ::= sequence[char] \times char^* \times SExpr^* \times Nat \times (Nat \to SExpr)$$

As for ComLisp, an evaluation semantics for SIL statements is defined as the smallest relation $\Gamma \vdash s : cmd \to q$ satisfying the set of rules given for the language constructs. The relation states that executing the statement cmd in state s and SIL procedure environment Γ (a list of procedure declarations) is defined, terminates, and results in a new state q. As for ComLisp, the semantics of a SIL program is its I/O behavior:

$$P_{\text{sem}_{\text{SIL}}}(p)(is, ol) ::= \exists q.\ (\Gamma \vdash init : s \to q)\ \wedge\ (q_{\text{output}} = ol)$$

where the initial state is defined by $init ::= (is, [], [NIL, \ldots, NIL], 0, \lambda n.NIL)$.

2.3 PVS Formalization of the Languages

Abstract syntax, static and dynamic semantics of the languages have to be formalized in the PVS specification language. The language is based on classical higher-order logic with a rich type system including dependent types. In addition, the PVS system provides an interactive proof checker that has a reasonable amount of theorem proving capabilities. A strategy language enables to combine atomic inference steps into more powerful proof strategies allowing to define reusable proof methods.

1. Abstract Syntax: the PVS abstract data type (ADT) construct is used. ComLisp forms, for example, are defined by an ADT, where for each kind of form there exists a corresponding constructor. For ADT definitions in PVS, a large theory is automatically generated including induction and reduction schemes for the ADT, termination measures, and a set of axioms stating that the data type denotes the initial algebra defined by the constructors. Note that the

formalizations make heavily use of library specifications. However, a lot of new types, functions, and predicates must be added for the specifications, as well as lemmas for their useful properties (which have to be proved).
2. Static Semantics: the well-formedness predicates must be formalized. Since each function must be total in PVS, a termination measure must be provided for the recursive definitions. We have specified the structural size of a ComLisp form using the reduction scheme from the ADT theory.
3. Dynamic Semantics: the rules must be represented in PVS. A set of structural rules is represented as an inductive PVS relation which combines all the rules in one single definition $E(\Gamma)(s, e, v, q, N)$ which denotes $\Gamma \vdash s : e \to (v, q)$. Free logical variables in the rules are existentially quantified in the corresponding PVS relation. In general, properties about inductive relations can be proved by rule induction. Here, the definition of relation E has an additional counter parameter N to formulate an induction principle needed for the proof for the selected notion of correctness (see Sect. 4). N is decreased when entering the body of a function or while loop, since in this case the forms in the antecedents of the corresponding rules are not structurally smaller, and left unchanged otherwise.

3 Compiling ComLisp to SIL

The compilation from ComLisp to SIL generates code according to the stack principle and translates parameter passing to statements which access the data stack. For a given expression e, a sequence of SIL instructions is generated that computes its value and stores it at the top of the stack (relative position k in the current frame). The parameters $x_1, \ldots, x_n$ of a function are stored at the bottom of the current frame (at relative positions $0, \ldots, n-1$). A SIL function call $\mathit{fcall}(h, i)$ increases the frame pointer *base* by i which is reset to its old value after the call and local variables introduced by *let* are represented within the current frame. For each syntactical ComLisp category, a compiling function is specified.

- $\mathcal{C}_{\text{form}}(e, \gamma, \rho, k)$ is defined inductively on e. It takes a form e, a global environment γ (a list of identifiers), a compile time environment ρ (an association list which associates relative positions in the current stack frame with local variables), and a natural number k (denoting the current top of stack) and produces a SIL statement (definition omitted).
- A function definition is compiled by compiling the body in a new environment (where the formal parameters are associated with relative positions $0, \ldots, n-1$) with the top of stack set at position n. Finally, the current stack frame has to be removed, leaving only the result on top (achieved by a copy instruction from position n to 0).

  $$\mathcal{C}_{\text{def}}(h(x_1, \ldots, x_n) \leftarrow e)(\gamma) ::= h \leftarrow \mathit{sq}(\mathcal{C}_{\text{form}}(e, \gamma, [x_i \leftarrow i-1], n), \mathit{copy}(n, 0))$$

- The compilation functions for function environments $\mathcal{C}_{\text{defs}}(\Gamma)(\gamma)$ and ComLisp programs $\mathcal{C}_{\text{prog}}(p)$ are straightforward and omitted here.

4 Correctness of the Compilation Process

An appropriate notion of correct compilation for sequential imperative languages on a concrete target processor must take the finite resource limitations of the target architecture into account. The notion of correctness used in *Verifix* is the preservation of the observable behavior up to resource limitations. In our case correctness of the compilation process is stated as follows: for any well-formed ComLisp program p, whenever the semantics of the compiled program is defined for some input stream is and output list ol, this is also the case for p for the same is and ol:

Theorem 1 (Correctness of Program Compilation).
$\forall p, is, ol.\ wf_{\text{program}}(p) \Rightarrow (P_{\text{sem}_{\text{SIL}}}(\mathcal{C}_{\text{prog}}(p))(is)(ol) \Rightarrow P_{\text{sem}_{\text{CL}}}(p)(is)(ol))$

Unfolding $P_{\text{sem}_{\text{SIL}}}$ and $P_{\text{sem}_{\text{CL}}}$, the semantics of forms and corresponding SIL statements have to be compared. In particular, this requires relating source and target language states. ComLisp forms denote state transformers transforming a state into a result value and a result state (if defined) $\sigma \rightarrow_e (v, \sigma')$. On the other hand, SIL statements denote ordinary state transformers $s \rightarrow_s s'$. Two relations are required: one relation ρ_{in} relates ComLisp input states σ with SIL states s, while the other relation ρ_{out} relates ComLisp output states (v, σ') with SIL states s'. Figure 1 illustrates the correctness property for forms by means of a commuting diagram. The relations are parameterized with a list of global

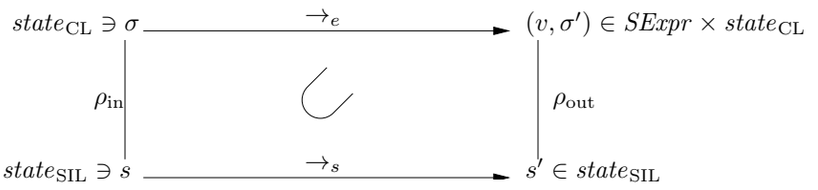


Fig. 1. Correctness property for the compilation of ComLisp forms

variables γ, the local compile time environment ρ, and the current top of stack position k. Relation ρ_{in} distinguishes between local and global variables. The relative address for variables for which ρ is defined is given by $\rho(x)$, while the address of the global variables in γ is given by $\gamma(x)$. Relation ρ_{out} additionally assumes that the final value v is available at the stack top (relative address k). In addition, it is required that the input streams and the output lists of σ and s correspond. The data representation relations are defined as follows:

$\rho_{\text{in}}(\gamma, \rho, k)(\sigma, s) ::=$
$[\forall x \in dom(\rho).\ (\rho(x) < k)\ \wedge\ (\sigma(x) = s_{\text{local}}(s_{\text{base}} + \rho(x)))] \wedge$
$[\forall x \in \gamma.\ (\gamma(x) < |s_{\text{global}}|)\ \wedge\ (\sigma(x) = s_{\text{global}}(\gamma(x)))] \wedge$
$(s_{\text{input}} = \sigma_{\text{input}})\ \wedge\ (s_{\text{output}} = \sigma_{\text{output}})$
$\rho_{\text{out}}(\gamma, \rho, k)(v, \sigma', s') ::= (s'_{\text{local}}(s'_{\text{base}} + k) = v)\ \wedge\ (\rho_{\text{in}}(\gamma, \rho, k)(\sigma', s'))$

In order to state the correctness property for the compilation of forms two additional invariants are required:

1. The first invariant ($source_invar?(\zeta,\gamma)(\sigma,\sigma')$) relates ComLisp input and output states. It assures that identifiers not belonging to ζ or γ (the local and global identifier lists) do not alter their values.
2. The second one ($invar?(\rho,k)(s,s')$) relates input SIL states s with output SIL states s'. It states that
   (a) the frame pointers of s and s' are identical.
   (b) the contents of all stack cells with addresses not within the range of the local environment ρ do not change from s to s'. In particular, this includes all stack cells below the current stack frame.

   This property is required to ensure that for function and operator calls the computed values of the arguments are still available (and not overwritten) when the operator is applied or the function body is executed.

All ingredients have now been collected to state the correctness property for the translation of forms. The diagram in Fig. 1 has to commute in the sense of preservation of partial program correctness. The property states that if the function environment and the ComLisp form is well-formed, the compile time environment ρ is injective and its domain corresponds to the local variable list ζ, the initial ComLisp and SIL states are related by ρ_{in} and the code resulting from compiling form e transforms SIL state s into s', then there exists a value v and ComLisp state σ' such that e evaluates in state σ to (v,σ') and the final ComLisp and SIL states are related by ρ_{out} and the target states and source states invariants hold:

Definition 1 (Correctness Property for Form Compilation).
$$
\begin{aligned}
&correct_prop(\Gamma,\gamma,\zeta,\rho,k)(e) ::= \\
&\forall\sigma,s,s'.\ wf_{\text{proc}}(\Gamma,\gamma)\ \wedge\ wf(e,\zeta,\gamma,\Gamma)\ \wedge\ injective?(\rho)\ \wedge\ (dom(\rho)=\zeta)\ \wedge \\
&\quad \rho_{\text{in}}(\gamma,\rho,k)(\sigma,s)\ \wedge\ (\mathcal{C}_{\text{defs}}(\Gamma)(\gamma)\vdash s:\mathcal{C}_{\text{form}}(e,\gamma,\rho,k)\to s') \\
&\quad\Rightarrow \exists v,\sigma' : (\Gamma\vdash\sigma : e\to(v,\sigma'))\ \wedge\ \rho_{\text{out}}(\gamma,\rho,k)(v,\sigma',s')\ \wedge \\
&\qquad invar?(\rho,k)(s,s')\ \wedge\ source_invar?(\zeta,\gamma)(\sigma,\sigma')
\end{aligned}
$$

The main obligation is to prove that this property holds for each kind of form:

Theorem 2 (Correctness of Form Compilation).
$\forall e,\Gamma,\gamma,\zeta,\rho,k.\ correct_prop(\Gamma,\gamma,\zeta,\rho,k)(e)$

In the PVS formalization, the correctness property has an additional counter argument N according to the inductive relations defining the semantics. This additional argument is required here since we prove that the target semantics implies the source semantics but the compilation is defined structurally on the source language. If we would prove the other way round, rule induction (without a counter argument) would suffice. The PVS proof of this theorem is done by measure induction (a variant of well-founded induction) using the lexicographic combination of the counter N and the structural size of form e as termination measure. This measure ensures that for each kind of form the induction hypothesis is applicable. To suitably manage the complexity of this proof, for each kind

of form a separate compilation theorem is introduced. The proof of Theorem 2 is then carried out by case analysis and application of the compilation theorems.

Most of the proofs of the compilation theorems follow a similar scheme according to the structure of the correctness property (see Definition 1):

1. First, definitions must be unfolded and the SIL statement which results from compiling the ComLisp form must be "executed" symbolically according to the operational SIL semantics.
2. The induction hypothesis (stated as a precondition in the compilation lemmas) must be instantiated.
3. Instantiations for the result value v and result state σ' (existentially quantified variables) of the ComLisp form must be found.
4. The consequent part of the formula must be proved. This reduces to showing four properties:
   (a) show that form e evaluates to the instantiated value and result state.
   (b) show with the help of precondition ρ_{in} that the output source and target states are related by ρ_{out} (Note that ρ_{out} is defined by means of ρ_{in}).
   (c) show that the target state invariant holds.
   (d) show that the source state invariant holds.

PVS strategies have been defined for some of the cases of the general scheme. These strategies enable the (nearly) automatic discharge of the respective cases. The proofs of most of the compilation lemmas are relatively straightforward and follow directly the scheme. However, some of the compilation theorems are tedious, in particular the theorems for function call, let-form, and *list*$*$. They make use of an additional lemma which relates sequences of ComLisp forms with SIL statement sequences. Due to lack of space we cannot go into the details of the proofs. All the proofs have been completely accomplished using PVS.

Statistics

We present some statistics concerning the formalization and verification effort for this compilation step. Table 1 summarizes the results. First of all, we have extended the built-in PVS library with additional functions and properties for lists, and with a new theory for association lists (finite maps). This library has already been reused for other verification tasks. There are 7 additional PVS theories with 621 lines of PVS specification code (LOC), 139 obligations to prove including all type correctness conditions generated by the system. These obligations are proved interactively by invoking 1048 proof steps. The specifications of the languages ComLisp and SIL including the definition of s-expressions and corresponding unary and binary operators involve 7 theories. Not surprisingly, the most effort lies in the verification of the compiling specification: 30 proof obligations (mainly the compiling theorems) have been proved in more than 1600 proof steps. Most work has been put into the verification of the compilation theorems for function call, *let*, and *list*$*$. Although strategies for parts of the proofs have been developed, the number of manual steps is quite high and shows that this verification task is by no means trivial.

It is hard to give an estimation of the amount of work invested in the final verification, since we started the verification on a smaller subset of ComLisp in order to experiment with different styles of semantics and find the necessary invariants, and then incrementally extended this subset and tried to rerun and adapt the already accomplished proofs. A coarse estimation of the total formalization and verification effort required for the compiling specification for all 4 compilation phases is about 3 person-years.

Table 1. Formalization and verification statistics

	PVS theories	LOC	proof obligations	proof steps
spec. of languages	7	759	139	575
compiling specification	1	122	36	95
compiling verification	1	219	30	1617
list, alist library	7	621	139	1048
	16	1721	344	3335

Related Work

Verification of compiler correctness is a much-studied area starting with the work by McCarthy and Painter in 1967 [13], where a simple compiler for arithmetic expressions has been proved correct. Many different approaches have been taken since then, usually with mechanized support to manage the complexity of the specifications and the proofs, for example [17, 12, 2, 14, 4, 1]. Most of the approaches only deal with the correctness of the compiling specification, while the approach taken in the *Verifix* project also takes care of the implementation verification, even on the level of binary machine code. Another difference of our approach is that we are concerned with the compilation of "realistic" source languages and target architectures. A ComLisp implementation of the ComLisp compiler as well as a binary Transputer executable is available.

Notable work in this area with mechanized support is CLInc's verified stack of system components ranging from a hardware-processor up to an imperative language [14]. Both the compiling verification and the high-level implementation (in ACL2 logic which is a LISP subset) have been carried out with mechanized support using the ACL2 prover. Using our compiler, correct binary Transputer code could be generated.

The impressive VLISP project [10] has focused on a correct translation for Scheme. However, although the necessity of also verifying the compiler implementation has been expressed this has explicitly been left out. Proofs were accomplished without mechanized support.

P. Curzon [4] considers the verification of the compilation of a structured assembly language, Vista, into code for the VIPER microprocessor using the HOL system. Vista is a low-level language including arithmetic operators which correspond directly to those available on the target architecture.

The compilation of PROLOG into WAM has been realized through a series of refinement steps and has been mechanically verified using the KIV system [18]. A (small-step) ASM semantics is used for the languages.

5 Concluding Remarks

In this paper we have reported on an ongoing effort in constructing a correct bootstrap compiler for a subset of Common Lisp into binary Transputer code. We have focused on the formal, mechanically supported verification of the compiling specification of the first compilation phase. The verification of the second phase, the translation from SIL to C^{int}, where s-expressions and their operators are implemented in linear memory (classical data and operation refinement), is also completed. Current work is concerned with the verification of the compiler back-end, namely, the compilation from C^{int} into abstract Transputer assembler code TASM. The standard control structures of C^{int} must be implemented by conditional and unconditional jumps, and the state space must be realized on the concrete Transputer memory. Hence, this step is again a data refinement process to be verified. The verification of the last compilation phase, where abstract Transputer assembler is compiled into binary Transputer code (TC) has already been accomplished following approved verification techniques [15]: starting from a (low-level) base model of the Transputer, where programs are a part of the memory, a series of abstraction levels is constructed allowing different views on the Transputer's behavior and the separate treatment of particular aspects.

We have demonstrated that the formal, mechanized verification of a non-trivial compiler for a (nearly) realistic programming language into a real target architecture is feasible with state-of-the-art prover technology.

Acknowledgements

The construction of the initial correct ComLisp compiler is joint work with the project partners from the university of Kiel. We thank them for their constructive collaboration and many discussions on this subject. The constructive criticism and suggestions provided by the anonymous referees have been a great help to improve this paper.

References

1. E. Börger and W. Schulte. Defining the Java Virtual Machine as Platform for Provably Correct Java Compilation. In *23rd Int. Symposium on Mathematical Foundations of Computer Science*, volume 1450 of *LNCS*. Springer, 1998.
2. M. Broy. Experiences with Software Specification and Verification using LP, the Larch Proof Assistant. Technical report, Digital Systems Research Center, 1992.
3. L.M. Chirica and D.F. Martin. Toward Compiler Implementation Correctness Proofs. *ACM Transactions on Programming Languages and Systems*, 8(2):185–214, April 1986.

4. Paul Curzon. The Verified Compilation of Vista Programs. Internal Report, Computer Laboratory, University of Cambridge, January 1994.
5. Axel Dold. *Formal Software Development using Generic Development Steps*. Logos-Verlag, Berlin, 2000. Dissertation, Universität Ulm.
6. W. Goerigk, A. Dold, T. Gaul, G. Goos, A. Heberle, F. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Ruess, and W. Zimmermann. Compiler Correctness and Implementation Verification: The Verifix Approach. In *Proceedings of the Poster Session of CC'96 - International Conference on Compiler Construction*. ida, 1996. TR-Nr.: R-96-12.
7. Wolfgang Goerigk, Thilo Gaul, and Wolf Zimmermann. Correct Programs without Proof? On Checker-Based Program Verification. In *Proceedings ATOOLS'98 Workshop on "Tool Support for System Specification, Development, and Verification"*, Advances in Computing Science, Malente, 1998. Springer Verlag.
8. Wolfgang Goerigk and H. Langmaack. Compiler Implementation Verification and Trojan Horses. In D. Bainov, editor, *Proc. of the 9th Int. Colloquium on Numerical Analysis and Computer Sciences with Applications, Plovdiv, Bulgaria*, 2000.
9. G. Goos and W. Zimmermann. Verification of Compilers. In B.Steffen E.-R. Olderog, editor, *Correct System Design*, volume 1710 of *LNCS*, pages 201–230. Springer-Verlag, 1999.
10. J. D. Guttman, L. G. Monk, J. D. Ramsdell, W. M. Farmer, and V. Swarup. A Guide to VLISP, A Verified Programming Language Implementation. Technical Report M92B091, The MITRE Corporation, Bedford, MA, September 1992.
11. Ulrich Hoffmann. *Compiler Implementation Verification through Rigorous Syntactical Code Inspection*. PhD thesis, Technische Fakultät der Christian-Albrechts-Universität zu Kiel, Kiel, 1998.
12. J.J. Joyce. A Verified Compiler for a Verified Microprocessor. Technical Report 167, University of Cambridge, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, England, March 1989.
13. J. McCarthy and J.A. Painter. Correctness of a Compiler for Arithmetical Expressions. In J.T. Schwartz, editor, *Proceedings of a Symposium in Applied Mathematics, 19, Mathematical Aspects of Computer Science*. American Mathematical Society, 1967.
14. J S. Moore. A Mechanically Verified Language Implementation. *Journal of Automated Reasoning*, 5(4), 1989.
15. Markus Müller-Olm. *Modular Compiler Verification*, volume 1283 of *LNCS*. Springer Verlag, Berlin, Heidelberg, New York, 1997. PhD Thesis.
16. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
17. W. Polak. *Compiler Specification and Verification*, volume 124 of *LNCS*. Springer-Verlag, 1981.
18. Gerhard Schellhorn. *Verifikation abstrakter Zustandsmaschinen*. PhD thesis, Universität Ulm, 1999.

Beyond Regular Model Checking
(Extended Abstract)*

Dana Fisman and Amir Pnueli

Dept of Computer Science, The Weizmann Institute of Science, Rehovot, Israel.
{dana,amir}@wisdom.weizmann.ac.il

Abstract. In recent years, it has been established that *regular model checking* can be successfully applied to several parameterized verification problems. However, there are many parameterized verification problems that cannot be described by regular languages, and thus cannot be verified using regular model checking. In this study we try to practice symbolic model checking using classes of languages more expressive than the regular languages. We provide three methods for the uniform verification of non-regular parameterized systems.

1 Introduction

During the last two decades, several formal methods have been developed to answer the *verification problem* of finite-state systems. The verification problem asks the question of whether a given reactive system is correct relative to some specification. Although many interesting concurrent programs are in fact finite state, they are often given semantically in terms of a parameter n, representing the number of concurrent processes. Such a schematic program really represents an infinite family of uniformly defined programs. Such programs are often referred to as *parameterized systems*. A challenging problem is to provide methods for the *uniform* verification of parameterized systems, i.e., proving correctness for all possible programs obtained by instantiating the parameter. We refer to this problem as the *parameterized verification problem*. In 1986 Apt and Kozen proved that, in general, the parameterized verification problem is undecidable, even when each instance is finite-state [3]. However, for specific families the problem may be solvable.

Model Checking (MC) is an automatic technique for answering the verification problem. In this framework, specification are usually expressed by a propositional temporal logic and programs are modeled by state-transition systems. The model checking procedure performs an exhaustive search of the state space of the system to determine whether the system satisfies the specification. The use of an exhaustive state-space exploration limits the application of model checking to finite-state systems.

* This work was supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082), and carried out at the John von Neumann Minerva Center for the Verification of Reactive Systems.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 156–170, 2001.
© Springer-Verlag Berlin Heidelberg 2001

However, it is possible to perform a state-space exploration of infinite-state systems as well, by using an implicit representation for sets of states. The framework of model checking where sets of states are represented implicitly using some symbolic representation is known as *symbolic model checking (SMC)* [11]. Symbolic model checking can be applied to infinite-state systems, although in this case, the termination of the procedure is not guaranteed.

Regular model checking is an application of symbolic model checking where regular expressions are used to represent symbolically sets of states [21]. Regular model checking can be applied to any verification problem that is expressible using regular languages. Such an application is successful if the regular model checking procedure has terminated. In recent years, it has been established that regular model checking can be successfully applied to several types of parameterized verification problems.

However, many interesting parameterized systems cannot be handled by regular model checking since the class of regular languages is not strong enough to express them. An example for such a verification problem is the Peterson algorithm for mutual exclusion among n processes [23]. The existence of such examples, is the main motivation for this study.

In this study we try to practice symbolic model checking using as symbolic representation classes of languages more expressive than the regulars. As a first attempt, we use context-free languages at the last step of the symbolic model checking procedure, while regular languages are used in the procedure until this last step. This seemingly slight change, already enables us to verify mutual exclusion for the Peterson algorithm.

By carefully examining the model checking procedure, one can compile a list of the requirements a class of languages must meet in order to be *adequate* for symbolic model checking [21]. Such a list consists of several operations the class must be effectively closed under, and several questions that must be effectively decidable for the class. We recognize that the class of languages accepted by deterministic pushdown automata $\mathcal{L}_{DPDA}$, meets all requirements but two: the class is not closed under projection and there is no known efficient algorithm to decide equivalence. We thus direct our effort to find a class, which is a subset of the class $\mathcal{L}_{DPDA}$, and possesses efficient algorithms for computing projection and deciding equivalence. This class must also satisfy the rest of the requirements.

We succeed to define a sub-class of $\mathcal{L}_{DPDA}$, which we denote $\mathcal{L}_{DPDA-M}$, for which there exists a semi-algorithm[1] to compute projection. In addition, there exists an efficient algorithm to answer the equivalence problem for this class. This class also satisfies all other requirements. Thus, we establish a class, which is more expressive than the class of regular languages, and yet is adequate for symbolic model checking. The Peterson example can be symbolically model checked, using languages in this class. Note that the notion of adequacy achieved here is weaker than the one introduced in [21] because termination failure can

[1] By semi-algorithm we mean a computational procedure that is not guaranteed to halt, but is guaranteed to give a correct answer in all cases at which it does halt. In practice, we use semi-algorithms by running them up to a prescribed time limit.

also occur in the computation of projection. However, due to the general undecidability of the problem, a semi-algorithm is the best we can hope for in any case.

Recall that the standard fix-point computation in the symbolic model checking procedure is not guaranteed to terminate when applied to infinite-state systems. This difficulty must be addressed when considering a class of languages to be adequate for symbolic model checking. In order for such a class to be *practically* adequate for symbolic model checking, it must also provide means to tackle this difficulty. Indeed, for regular model checking, many techniques to overcome this problem have been developed (see related work section). The common idea behind these techniques lies in calculating the effect of taking an arbitrary number of system-transitions in one step, often refer to as calculating meta-transitions or "accelerations".

For a special case of the class $\mathcal{L}_{DPDA-M}$ there exists an algorithm to compute projection. In addition, all techniques developed for calculating meta-transitions for regular model checking, can be applied to this (sub-)class. This class is therefore *practically* adequate for symbolic model checking. The Peterson example can also be verified using languages in this class.

Due to space limitation, further details and proofs are omitted, these can be found in the full version of the paper.[2]

Related Work. Regular model checking has been advocated by [21] and [30] as a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. The use of regular languages to express state properties goes back to [13]. The problem of calculating meta-transitions is one of the most laborious problems in this field of research. It has been thoroughly researched [1,2,18], resulting in several corrective techniques, such as acceleration [24,4], calculation of the transitive closure [20,9] and widening [22,9,15].

The study of model checking pushdown systems has recently been receiving growing attention [12,5,10,6,7,30,16]. Nevertheless, these efforts do not pertain to this study for two reasons: first, we use the pushdown automata to represent the set of states, while the above work considers the pushdown automata as the system being analyzed. Furthermore, a fundamental difference is that all systems previously considered share a common characteristic: a regular language represents their state space, whereas our main interest is systems whose state space cannot be represented by a regular language.

There has been other works also pursuing symbolic representations which are more expressive than regular languages. Studies by Boigelot and Wolper [29] and Comon and Jurski [14] give symbolic representations for configurations of systems with counters.

Perhaps the work most related to this research is by Bouajjani and Habermehl [8]. They define a symbolic representation, denoted CQDD, which is an extension of the QDDs defined by [5]. While QDDs are finite automata, CQDDs are a combination of restricted finite automata and linear constrains on the number of occurrences of symbols. The CQDDs are a symbolic representation which is more expressive than regular languages. As an example, they show that CQDDs accept the language $a^n b^m a^n b^m$. However, to our understanding, the CQDDs are not strong enough to express the Peterson example, which we were able to verify using all our methods.

[2] the full version of the paper can be found in `www.wisdom.weizmann.ac.il/~dana`

2 Preliminaries

Definition 1. Let Σ be an alphabet. A *bi-letter* over Σ is an element of $\Sigma \times \Sigma$. A *bi-word* over Σ is a string of bi-letters over Σ (i.e., an element of $(\Sigma \times \Sigma)^*$). A *bi-language* over Σ is a set of bi-words over Σ (i.e., a subset of $(\Sigma \times \Sigma)^*$). We use the notations $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right]$ to denote respectively the bi-letter (a, b) and the bi-word $\left[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} a_n \\ b_n \end{smallmatrix}\right]$ where $u = a_1 a_2 ... a_n$ and $v = b_1 b_2 ... b_n$.

Given a bi-language L over Σ, we denote by $L\Downarrow_1$ ($L\Downarrow_2$) the *projection* of L on the first (respectively, second) coordinate. Given a language L over Σ, we denote by $L \times \Sigma^*$ the language $\left\{ \left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right] \ : \ w \in L,\ u \in \Sigma^*,\ |u| = |v| \right\}$, referred to as *left lifting*. Similarly, we define *right lifting* and use the notation $\Sigma^* \times L$. We use the standard notations $\overline{L}$ and $L_1 \cap L_2$ to denote the complement of a language and the intersection of two languages respectively.

Definition 2. A *system* is a quadruple $(\Sigma, \chi, \Theta, \rho)$ where

- Σ is a finite alphabet.
- $\chi \subseteq \Sigma^*$ is a language over Σ, denoting the set of *states*.
- $\Theta \subseteq \chi$ is a language over Σ, denoting the set of *initial states*.
- $\rho \subseteq \chi \times \chi$ is a bi-language over Σ, denoting the *transition relation*.

2.1 Symbolic Model Checking and Regular Model Checking

The *verification problem* for a system M and a property φ is to decide whether φ holds over all computations of M. *Model checking* is an automatic technique for answering the verification problem. In *symbolic model checking (SMC)*, some symbolic representation $\mathcal{C}$ is used to represent sets of states. In the framework of symbolic model checking using the representation $\mathcal{C}$, the system and the property to be verified are represented as expressions in $\mathcal{C}$: The system is defined using $\mathcal{C}$-expressions to describe its components, which are essentially sets of states or relations over sets of states. The invariance property is defined using some expression in $\mathcal{C}$ to describe the sets of states satisfying it. Then, the model checking procedures operate by manipulating sets of states (instead of individual states) through operations on the expressions of $\mathcal{C}$. Symbolic model checking can be applied to infinite-state systems, although in this case, the termination of the procedure is not guaranteed. We refer to this problem as the *convergence problem*.

Regular model checking [21,30] is an application of symbolic model checking, where regular expressions are used as the (symbolic) assertional language for describing sets of states. Using this framework, the system's components, and the property to be verified are defined by means of regular expressions. Usually, we assume a given alphabet Σ to represent a local state. The set of global states χ, the set of initial states Θ and the property φ, are specified by a regular expression (defining a regular language) over Σ. The transition relation ρ is specified by a regular expression using bi-letters over Σ (sometimes referred to as a bi-regular expression).

Example 1. (TOKEN-STRING)
Consider an array of processes that pass a token from left to right. We define the alphabet $\Sigma = \{1, 0\}$ to denote the local state of a process, i.e., that the process has or has not the token. A global state of a system consisting of n processes is defined by a word of length n, each letter describes the state of one process. The set of global states χ is defined to be the language of all words of positive length, given by the regular expression $(0 + 1)^+$. The set of initial states Θ, is given by the regular expression 10^*, indicating that the leftmost process holds the token. The transition relation ρ can be specified by the bi-regular expression $\left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^* \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] \left(\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] + \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right)^*$ defined over $\Sigma = \{0, 1\}$. Alternatively, we can specify the transition relation by the length-preserving rewrite rule $x\ 1\ 0\ y \rightarrow x\ 0\ 1\ y$ (where x and y are arbitrary words over $\{0, 1\}^*$). The invariance property stating that there is exactly one token at all times, can be given by $\varphi = 0^*\ 1\ 0^*$.

3 Adequacy of Classes of Automata for SMC

Regular model checking is an important application of symbolic model checking which enables the verification of several classes of parameterized and infinite-state systems. Nevertheless, there are instances where a regular language is not expressive enough to describe the system or the property at hand. Our aim is to find a class of languages that is more expressive than the regular languages, yet is adequate for symbolic model checking. Naturally, we are looking for the largest class of languages, which are still amenable to symbolic model checking.

The *rich-language symbolic model checking methodology* described in [21] lists a set of minimal requirements from an assertional language, in order for it to be adequate for symbolic model checking. We reformulate these requirements here in terms of classes of languages rather than assertions. We classify the languages involved in the symbolic model process, according to the operations applied to them. Our intention is to allow deployment of different classes of languages within the symbolic model checking process. We present *backward* and *forward model checking procedures* in terms of this classification.

Let $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ be three classes of languages. Let $M_\varphi \in \mathcal{M}$ and $A_\varphi \in \mathcal{A}$ be languages adequate for specifying the property to be verified. Let $A_\Theta \in \mathcal{A}$ and $M_\Theta \in \mathcal{M}$ be languages adequate for representing the initial state of the system. Let $R_\rho \in \mathcal{R}$ be a bi-language adequate for representing the transition relation of the system, augmented by the identity relation (idle transition). We will use the auxiliary languages $M_0, M_1, M_2, ... \in \mathcal{M}$ to represent system states. The following procedures describe backward and forward model checking:

Procedure *Backward MC*
  $M_0 := \overline{M_\varphi}$
  For $i = 0, 1, \ldots$ **repeat**
    $M_{i+1} := ((\Sigma^* \times M_i) \cap R_\rho)\Downarrow_1$
  until $M_{i+1} = M_i$
return $M_i \cap A_\Theta = \emptyset$

Procedure *Forward MC*
  $M_0 := M_\Theta$
  For $i = 0, 1, \ldots$ **repeat**
    $M_{i+1} := ((M_i \times \Sigma^*) \cap R_\rho)\Downarrow_2$
  until $M_{i+1} = M_i$
return $M_i \cap \overline{A_\varphi} = \emptyset$

The classes $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{A}$ are *adequate* for symbolic model checking, if the following requirements hold:

1. $\mathcal{R}$ is adequate for representing ρ, and either $\mathcal{M}$ and $\mathcal{A}$ are adequate for specifying φ and Θ respectively (for *Backward MC*) or $\mathcal{M}$ and $\mathcal{A}$ are adequate for specifying Θ and φ respectively (for *Forward MC*).
2. Either $\mathcal{M}$ is effectively closed under complementation (for *Backward MC*) or $\mathcal{A}$ is effectively closed under complementation (for *Forward MC*).
3. $\mathcal{M}$ is effectively closed under lifting.
4. $\mathcal{M}$ is effectively closed under intersection with $\mathcal{R}$.
5. $\mathcal{M}$ is effectively closed under projection.
6. Either $\mathcal{M}$ is effectively closed under intersection with $\mathcal{A}$ and emptiness is effectively decidable for $\mathcal{M}$, or $\mathcal{A}$ is effectively closed under intersection with $\mathcal{M}$ and emptiness is effectively decidable for $\mathcal{A}$.
7. Equivalence of two languages in $\mathcal{M}$ is effectively decidable.

Assuming that ρ always includes the identity relation (implementing the idling step), it is unnecessary to require closure under union (as required in [21]).

3.1 Meeting the Requirements

The following combinations appear to meet all the requirements:

1. Taking $\mathcal{M} = \mathcal{R} = \mathcal{A}$ to be the class of regular languages $\mathcal{L}_{FA}$. This leads to regular model checking.
2. Taking $\mathcal{M} = \mathcal{R} = \mathcal{L}_{FA}$ and letting $\mathcal{A}$ be the class of languages accepted by deterministic pushdown automata $\mathcal{L}_{DPDA}$. This combination is demonstrated in Section 4 by verifying Peterson's mutual exclusion algorithm for an arbitrary number of processes.

A more challenging step is considering $\mathcal{M}$ to be the $\mathcal{L}_{DPDA}$ class, leaving the $\mathcal{L}_{FA}$ class for $\mathcal{R}$ and $\mathcal{A}$. Requirement 3 is met as $\mathcal{L}_{DPDA}$ is closed under inverse homomorphism, which is a generalization of lifting. Requirements 2, 4 and 6 are met as $\mathcal{L}_{DPDA}$ is closed under complementation and intersection with a regular language, and the emptiness problem for $\mathcal{L}_{DPDA}$ is decidable. However, requirement 5 is not met: $\mathcal{L}_{DPDA}$ is not closed under projection. Requirement 7, decidability of the equivalence problem, was an open question until recently. In 1997 it was proven positively by Senizergues [26]. However, the algorithm he provides is not effective [28].

We thus concentrate our efforts to find a sub-class of $\mathcal{L}_{DPDA}$ which is adequate for symbolic model checking. Therefore, we must find effective algorithms to compute projection and to decide equivalence for languages in this class. In addition, this class must satisfy the rest of the closure and decidability properties discussed above. This is the topic of Section 5.

4 SMC Using Context-Free and Regular Languages

In this section we concentrate on the framework of symbolic model checking where the class $\mathcal{A}$ is chosen to be the $\mathcal{L}_{DPDA}$ class, while $\mathcal{M}$ and $\mathcal{R}$ are the $\mathcal{L}_{FA}$ class. If we choose to apply backward model checking we can describe the property to be verified by some language in the $\mathcal{L}_{DPDA}$ class, but then we have to compromise with a regular language for describing the initial condition. Alternatively, we can choose to apply forward model checking, which results in the initial condition being in $\mathcal{L}_{DPDA}$, while the property must be described by a regular language.

4.1 Peterson's Algorithm

We focus on the verification of the Peterson algorithm for mutual exclusion among N processes presented in the figure below.

The Peterson algorithm can be explained as follows. Each process $P[i]$ has a priority variable $y[i]$. The range of the priority variable are the numbers from 0 to $N-1$. In addition there are $N-1$ signature variables $s[1], s[2], \ldots s[N-1]$. The domain of the signature variables consists of the processes indices $1, 2, ..., N$. The variable $s[j]$ holds the signature (index) of the last process that received priority j. Assume process $P[i]$ has priority j. In order to increment its priority (to $j+1$), it must be either the process with the highest priority, or not the last who signed in the signature $s[j]$. A process can enter the critical section if it has the highest priority $(N-1)$ and it is not the last who signed in (in the signature of the highest priority, $s[N-1]$). When a process exits the critical section its priority is reset.

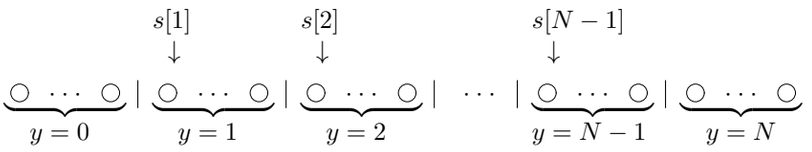
$$
\begin{array}{l}
N : \textbf{natural initially } N > 1 \\
y \ : \textbf{array } [1..N] \textbf{ of } [0..N-1] \textbf{ initially } y = 0 \\
s \ : \textbf{array } [1..N-1] \textbf{ of } [1..N]
\end{array}
$$

$$
\Big|\Big|_{i=1}^{N} P[i] :: \left[\begin{array}{l}
t : \textbf{integer} \\
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \textbf{Non-Critical} \\
\ell_2\text{: } \textbf{for } t := 1 \textbf{ to } N-1 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_3\text{: } (y[i], s[t]) := (t, i) \\
\ell_4\text{: } \textbf{await } s[t] \neq i \ \vee \ \forall j \neq i : y[j] < y[i]
\end{array}\right] \\
\ell_5\text{: } \textbf{Critical} \\
\ell_6\text{: } y[i] := 0
\end{array}\right]
\end{array}\right]
$$

Program Peterson(N).

4.2 Verification of Peterson's Algorithm

In order to apply our method, we must first model the system by languages as defined in Definition 2. We encode a state of the Peterson system as a word of the form



where all processes $P[i]$ within partition $k = 0, 1, \ldots, N-1$ have their priority variable $y[i]$ set to k. The leftmost process in partition k is the one signed in the signature $s[k]$. Processes in the rightmost partition are the ones which are in the critical section. Note that there are $N + 1$ partitions, separated by N border markers. The set of global states can thus be given by the language $\chi = \{w \in (\bigcirc + \mid)^* : \sharp(\mid, w) = \sharp(\bigcirc, w) > 1\}$. This is the language over the alphabet $\{\bigcirc, \mid\}$, each of whose words has an equal number of "$\mid$" and "$\bigcirc$". The initial state can be described by the regular expression $\Theta : \quad \bigcirc^* \mid^*$, in which all processes have their priority variable y set to 0 and thus are located in the first partition.

The transition relation ρ can be specified using five length-preserving rewrite rules: $\rho_1, \rho_2, \rho_3, \rho_4$ and ρ_{id} defined as follows:

$$
\begin{array}{lcll}
\rho_1 & : & x \bigcirc \mid y \mapsto x \mid \bigcirc y & \text{where } x \in \bigcirc^*, y \in (\mid + \bigcirc)^* \\
\rho_2 & : & x \mid \bigcirc \mid y \mapsto x \mid \mid \bigcirc y & \text{where } x \in (\mid + \bigcirc)^*, y \in \mid^* \\
\rho_3 & : & x \bigcirc \bigcirc \mid y \mapsto x \bigcirc \mid \bigcirc y & \text{where } x, y \in (\mid + \bigcirc)^* \\
\rho_4 & : & x \bigcirc \mapsto \bigcirc x & \text{where } x \in (\mid + \bigcirc)^* \\
\rho_{id} & : & x \mapsto x & \text{where } x \in (\mid + \bigcirc)^*
\end{array}
$$

The first rewrite rule states that a process can move unconditionally from the first partition to the second partition. This corresponds to increasing the priority variable y from 0 to 1. The second rule allows a single process within a partition can move to the next partition provided that all the partitions to it's right are empty. This corresponds to the situation where the process has the highest priority. The third rule allows a process to move to the next partition provided that there is a process to its left in the same partition. This corresponds to the situation where the process is not the last to sign in the signature corresponding to its current priority. The fourth rule describes an exit from the critical section,while the last rule captures the stuttering step.

Performing forward regular model checking, we obtain the following set of reachable states:

$$R : \qquad \bigcirc^* \mid (\bigcirc^+ \mid)^* \bigcirc^* \mid^* \quad + \quad \bigcirc^* \mid (\bigcirc^+ \mid)^* \mid^+ \bigcirc \mid^*.$$

The interpretation being: to the right of any empty partition ($\mid\mid$) there can be at most one process ($\bigcirc$). The negation of mutual exclusion is described by $x \bigcirc \bigcirc$, which represents the situation in which there are at least two processes in the critical section. This has a non-empty intersection with the set of reachable

states, given by: $\bigcirc^*\big|\,(\,\bigcirc^+\big|\,)^*\,\bigcirc^*\bigcirc\,\bigcirc$. Thus, we are unable to verify the Peterson algorithm using purely regular model checking.

However, when we express the negation of the property by an $\mathcal{L}_{DPDA}$-language which can be described by the intersection $(x\bigcirc\,\bigcirc\,)\cap\chi$, where χ requires an equal number of sticks and stones, we obtain an empty intersection. Thus, by mixing regular with context-free languages, we are able to verify mutual exclusion for the Peterson algorithm for an arbitrary number of processors, whereas using purely regular model checking, we failed to do so. An elaborated example and further details can be found in the full paper.[3]

5 SMC Using a Cascade Product 1DPDA ∘ DFA

In this section we seek to find a subset of $\mathcal{L}_{DPDA}$ which is adequate for symbolic model checking. Given a single state DPDA M we define the class $\mathcal{L}_{DPDA-M}$. We provide a semi-algorithm for computing projection for this class, and an efficient algorithm to decide equivalence. This class, in addition, is closed under all basic operations required in order to be adequate for symbolic model checking.

5.1 The Classes $\mathcal{L}_{DPDA-M}$

We concentrate on deterministic pushdown automata with no epsilon transitions, DPDAs [19]. We often consider single-state DPDAs with an empty set of accepting conditions, to which we refer as a 1DPDA. A 1DPDA $M=\langle\Sigma,\{q\},q,\Gamma,\bot,\Delta,\emptyset\rangle$ can be represented by the quadruple $\langle\Sigma,\Gamma,\bot,\widehat{\Delta}\rangle$ where $\widehat{\Delta}:\Sigma\times\Gamma\to Com(\Gamma)$. The following definitions are needed for the sequel.

Definition 3. (*cascade product*)
Let $R=\langle V\times\Gamma,S,s_0,\delta,F\rangle$ be a DFA and $\phi:V\to\Sigma$ a substitution mapping each letter of V to a letter in Σ. The *cascade product* $M\circ_\phi R$ is the DPDA $\langle V,S,s_0,\Gamma,\bot,\rho,F\rangle$ where $\rho(s,\sigma,z)=(\delta(s,\langle\sigma,z\rangle),\Delta(\phi(\sigma),z))$.

Definition 4. (*stack-consistent with M*)
A DPDA A is said to be *stack-consistent with M* (*M-consistent*), if there exist a substitution $\phi:V\to\Sigma$ and a DFA $R=\langle V\times\Gamma,S,s_0,\delta,F\rangle$ such that $A=M\circ_\phi R$. We say that A is *M-consistent w.r.t* (with repect to) ϕ.

Let $A=M\circ_\phi R$. A run π_A of A on the word $w=\sigma_1\sigma_2\cdots\sigma_n$ can be decomposed into two runs: a run π_M of M on $\phi(w)$, and a run π_R of R on w:

$$\begin{array}{lccccccc}
\pi_A: & (s_0,\gamma_0) & \overset{\sigma_1/x_1}{\vdash\!\!\!\!-\!\!\!\!-} & (s_1,\gamma_1) & \overset{\sigma_2/x_2}{\vdash\!\!\!\!-\!\!\!\!-} & \ldots & \overset{\sigma_n/x_n}{\vdash\!\!\!\!-\!\!\!\!-} & (s_n,\gamma_n)\\
\pi_M: & \gamma_0 & \overset{\phi(\sigma_1)/x_1}{\vdash\!\!\!\!-\!\!\!\!-} & \gamma_1 & \overset{\phi(\sigma_2)/x_2}{\vdash\!\!\!\!-\!\!\!\!-} & \ldots & \overset{\phi(\sigma_n)/x_n}{\vdash\!\!\!\!-\!\!\!\!-} & \gamma_n\\
\pi_R: & s_0 & \overset{\sigma_1,\top(\gamma_0)}{\vdash\!\!\!\!-\!\!\!\!-} & s_1 & \overset{\sigma_2,\top(\gamma_1)}{\vdash\!\!\!\!-\!\!\!\!-} & \ldots & \overset{\sigma_n,\top(\gamma_{n-1})}{\vdash\!\!\!\!-\!\!\!\!-} & s_n
\end{array}$$

[3] the full version of the paper can be found in `www.wisdom.weizmann.ac.il/~dana`

The decision upon the stack command at the i-th step x_i is governed only by the 1DPDA M, which is not aware of the state s_i of R. Yet, the DFA R can look at the top symbol on the stack of M ($\top(\gamma)$ denotes the top symbol of stack γ). Automata R and M move simultaneously: when A reads a letter $\sigma_i \in V$, the DFA R makes a move as if it was reading the pair $(\sigma_i, \top(\gamma_{i-1}))$, where γ_{i-1} is the current stack of M, while M makes a move as if it was reading $\phi(\sigma_i)$. The automaton A accepts a word w if, when the run terminates, R is in state $s_n \in F$.

An automaton $A = \langle V, S, s_0, \Gamma, \bot, \rho, F\rangle$ which is M-consistent w.r.t ϕ can be characterized by the pair $\langle R, \phi\rangle$ where R is the DFA such that $A = M \circ_\phi R$.

Definition 5. (*The M-stack consistent class, $\mathcal{L}_{DPDA-M}$*)
Given a 1DPDA M, we define the M-consistent class $\mathcal{L}_{DPDA-M}$ to be the class of languages that are accepted by some M-consistent DPDA.

The M-consistent automaton A can be viewed as a case in which the automaton has been decomposed into a stack-manipulator, which behaves exactly like M in its decisions about the stack transformations, and a finite-state controller R which affects the selection of the next state. Two M-consistent automata A_1 and A_2 share the same stack-manipulator and may differ in their respective finite-state controllers, as well as in their respective substitutions.

Claim. The class $\mathcal{L}_{DPDA-M}$ is *effectively closed* under complementation, lifting and intersection with a regular language.

Claim. Equivalence and Emptiness are *effectively decidable* for the class $\mathcal{L}_{DPDA-M}$.

5.2 Computing Projection

Viewing the claims above, for any 1DPDA M the class $\mathcal{L}_{DPDA-M}$ satisfies requirements 1,2,3,4,6 and 7. Thus, if the class is also effectively closed under projection, then it is adequate for symbolic model checking (in collaboration with the $\mathcal{L}_{FA}$ class). Below, we provide a semi-algorithm for computing projections.

Let A be a DPDA which is M-consistent w.r.t ϕ, and let R be A's characteristic DFA. For simplicity we assume the input alphabet of A is $\Sigma \times \Sigma$, where Σ is the input alphabet of M. Also, we assume A is M-consistent w.r.t $\phi = \Downarrow_2$ and we wish to calculate the projection of $\mathcal{L}(A)$ on its first coordinate. That is, we would like to compute another DPDA $\widetilde{A}$ over the alphabet Σ which is M-consistent w.r.t the identity relation and accepts the projection of A on its first coordinate.

We claim that procedure *Project* described below calculates the correct projection (unless it aborts). The procedure can be explained as follows. To simplify notations, we present ρ as a relation, a subset of $S \times \Sigma \times \Gamma \times Com(\Gamma) \times S$ instead of a function from $S \times \Sigma \times \Gamma$ to $S \times Com(\Gamma)$.

Procedure *Project*
Input: a DPDA $A = \langle \Sigma \times \Sigma, S, s_0, \Gamma, \bot, \rho, F \rangle$, a positive integer k
Output: a DPDA $\widetilde{A} = \langle \Sigma, \widetilde{S}, \widetilde{s_0}, \Gamma, \bot, \widetilde{\rho}, \widetilde{F} \rangle$

1. *Annotation*:
   $\widehat{A} = \langle \Sigma \times \Sigma, \widehat{S}, \widehat{s_0}, \Gamma, \bot, \widehat{\rho}, \widehat{F} \rangle := Annotate(A, k)$
2. *Projection*:
   $\ddot{A} := \langle \Sigma, \ddot{S}, \ddot{s_0}, \Gamma, \bot, \ddot{\rho}, \ddot{F} \rangle$ where $\ddot{S} = \widehat{S}$, $\ddot{s_0} = \widehat{s_0}$, $\ddot{F} = \widehat{F}$ and
   $\ddot{\rho} = \{(s, z_1, \sigma_1, x_1, s') \ : \ \exists z_2, \sigma_2, x_2 : \ (s, z_1, z_2, \sigma_1, \sigma_2, x_1, x_2, s') \in \widehat{\rho}\}$
3. *Determinization*:
   $\ddot{R} := \langle \Sigma \times \Gamma, \ddot{S}, \ddot{s_0}, \ddot{\delta}, \ddot{F} \rangle$ where $\ddot{\delta}(s, (\sigma, z)) = s' \iff \exists x : \ (s, z, \sigma, x, s') \in \ddot{\rho}$
   $\widetilde{R} = \langle \Sigma \times \Gamma, \widetilde{S}, \widetilde{s}, \widetilde{\delta}, \widetilde{F} \rangle := Subset_Construction(\ddot{R})$
Return $\widetilde{A} := M \circ_{id} \widetilde{R}$

In Phase 1 procedure *Project* calls procedure *Annotate* described below. Procedure *Annotate* guesses how the 1DPDA M will operate when looking at the first coordinate instead of the second. For each edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$[4] the procedure aims to find all possible stack letters z_1 and stack commands x_1 such that if M looks at the first coordinate then, when moving from state s to s', it will have z_1 on the top of the stack and decide on the stack command x_1.

For each state, it saves an information describing the difference between the actual stack of M and the "guessed stack" (the stack of M if it was looking at the first coordinate). For this it uses the notation $\langle \beta_1, \beta_2 \rangle$ with the intention that if the maximal common prefix of the actual stack and the guessed stack is w then the "guessed stack" is described by $w\beta_1$ and the actual stack is described by $w\beta_2$. An original state s of A may appear more than once in $\widehat{A}$, each time labeled with a different notation $\langle \beta_1, \beta_2 \rangle$.

Note that z_2 must be the top symbol of $w\beta_2$. The procedure will choose z_1 to be the top symbol of $w\beta_1$. The stack command x_1 is then determined by $\Delta(\sigma_1, z_1)$. Assuming that the difference between the guessed stack and actual stack in state s is $\langle \beta_1, \beta_2 \rangle$, and the stack commands for the guessed and actual stack are x_1 and x_2, we can compute the new difference $\langle \beta_1', \beta_2' \rangle$ for state s'. Let γ_1 and γ_2 be the result of applying x_1 and x_2 to the stack contents $w\beta_1$ and $w\beta_2$ respectively. Given w' is the maximal common prefix of γ_1 and γ_2, then $\beta_1' = \gamma_1 / w'$ and $\beta_2' = \gamma_2 / w'$ (where u/v denotes the right division of u by v).

To guarantee termination, the procedure uses the second parameter $-$ a bound $k \in N$. The procedure aborts if the length of either β_1 or β_2 exceeds k. It may be the case that the procedure decides to abort when exploring an *unreachable* edge. We say that an edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$ where s is annotated $\langle \beta_1, \beta_2 \rangle$ is unreachable if there is no prefix w such that $w\beta_2$ is in the reachable stack language of s. To avoid this situation, the procedure computes for each state s the reachable stack language of s. It uses a labeling function f_{reg} that labels each state with its reachable stack language, $\mathcal{L}(s)$. When exploring the edge (s, s') the procedure checks that $w\beta_2$ is in the reachable stack language of s, given by $r = f_{\mathsf{reg}}(s)$. The procedure updates the reachable stack language

[4] If $(s, z, x, \sigma, s') \in \rho$ we say that the edge (s, s') is labeled by (σ, z, x).

of s', $f\mathsf{reg}(s')$, to contain the set of words obtained by applying x_2 to a word in r whose top symbol is z_2.

Procedure *Annotate*

Input: $A = \langle \Sigma \times \Sigma, S, s_0, \Gamma, \bot, \rho, F \rangle$, a positive integer k
Output: $\widehat{A} = \langle \Sigma \times \Sigma, \widehat{S}, \widehat{s_0}, \Gamma, \bot, \widehat{\rho}, \widehat{F} \rangle$ where $\widehat{S} \subseteq S \times (\Gamma^{\leq k} \times \Gamma^{\leq k})$ and $\widehat{\rho} \subseteq S \times \Sigma^2 \times \Gamma^2 \times Com(\Gamma)^2 \times S$.

$\widehat{s_0} := (s_0, \langle \epsilon, \epsilon \rangle)$; $f\mathsf{reg}(\widehat{s_0}) := Compute_Stack_Language(A, s_0)$
$\widehat{\rho} := \emptyset$; $\widehat{S} := \{\widehat{s_0}\}$; $Q := \{\widehat{s_0}\}$
While $Q \neq \emptyset$ **do**
  Pick $\widehat{s} = (s, \langle \beta_1, \beta_2 \rangle) \in Q$
  For all $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta_1', \beta_2') \in ReachableTransitions$ s.t. $(s, \sigma_1, \sigma_2, z_2, x_2, s') \in \rho$ and $r = f\mathsf{reg}(s)$ **repeat**
    If $|\beta_1'| \geq k \ \vee \ |\beta_2'| \geq k$ **then abort**
    $\widehat{s'} := (s', \langle \beta_1', \beta_2' \rangle)$
    If $\widehat{s'} \notin \widehat{S}$ **then** $f\mathsf{reg}(\widehat{s'}) := r'$ **else** $f\mathsf{reg}(\widehat{s'}) := f\mathsf{reg}(\widehat{s'}) \cup r'$
    $\widehat{\rho} := \widehat{\rho} \cup \{(\widehat{s}, z_1, z_2, \sigma_1, \sigma_2, x_1, x_2, \widehat{s'})\}$; $\widehat{S} := \widehat{S} \cup \{\widehat{s'}\}$; $Q := Q \cup \{\widehat{s'}\}$
  end for all
  $Q := Q \setminus \{\widehat{s}\}$
end while
$\widehat{s_0} := \{(s, \langle \beta_1, \beta_2 \rangle) \ : \ s = s_0\}$; $\widehat{F} := \{(s, \langle \beta_1, \beta_2 \rangle) \ : \ s \in F\}$
end procedure

The complete relation between all involved components, is summarized as follows. Let $z_1, z_2 \in \Gamma$, $\sigma_1, \sigma_2 \in \Sigma$, $\beta_1, \beta_2, \beta_1', \beta_2' \in \Gamma^*$, $x_1, x_2 \in Com(\Gamma)$, $r, r' \in \mathcal{R}(\Gamma)$. $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta_1', \beta_2') \in ReachableTransitions$ if and only if the following holds:

1. $\exists w \in \Gamma^*$ such that $w\beta_2 \in r$
2. $z_1 = \top(w\beta_1)$ and $z_2 = \top(w\beta_2)$
3. $x_1 = \Delta(\sigma_1, z_1)$ and $x_2 = \Delta(\sigma_2, z_2)$
4. $\beta_1' = x_1(w\beta_1)/w'$ and $\beta_2' = x_2(w\beta_2)/w'$
   where $w' = max_common_prefix(x_1(w\beta_1), x_2(w\beta_2))$
5. $r' = \{x_2(uz_2) \mid uz_2 \in r\}$

The search is conducted on-the-fly starting at the initial state. For the initial state both the actual stack and guessed stack are $\bot$ and hence the initial state is annotated by $\langle \epsilon, \epsilon \rangle$. The reachable stack language of the initial state is computed by calling procedure *Compute_Stack_Languages* (see for example [17]). Then, for each state s labeled $f\mathsf{reg}(s) = r$ and annotated by $\langle \beta_1, \beta_2 \rangle$, and for each out-going edge (s, s') labeled by $(\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right], z_2, x_2)$, it annotates the state s' by $\langle \beta_1', \beta_2' \rangle$, updates $f\mathsf{reg}(s')$ to contain r' and adds the annotation z_1, x_1 to the edge, where $(r, r', z_1, z_2, x_1, x_2, \sigma_1, \sigma_2, \beta_1, \beta_2, \beta_1', \beta_2')$ satisfy the conditions of a reachable transition.

In Phase 2 procedure *Project* simply projects each edge on the first coordinate. From each edge on it eliminates the second component σ_2 of the bi-letter $\left[\begin{smallmatrix}\sigma_1\\ \sigma_2\end{smallmatrix}\right]$, the stack letter z_2 and the stack command x_2.

In Phase 3, as a first step the procedure extracts from the PDA $\ddot{A}$ the finite-state automaton $\ddot{R}$. Note that $\ddot{R}$ is non-deterministic (i.e. it is an NFA): from one state s there could be two outgoing edges (s, s') and (s, s'') with the same label. This non-determinism is the result of the projection performed in the previous phase. Hence, as a second step, the algorithm applies the subset construction and obtains the DFA $\widetilde{R}$.

In the full version of the paper, we also consider the simpler case where the cascade product DPDA$\circ$DFA is degenerate, i.e. the DFA does not look at the stack of the 1DPDA. In this case the language is given by a simple product of a DPDA and a DFA.

6 Conclusions and Future Work

Our research tried to give an answer to the verification problem of parameterized system, which cannot be answered using regular model checking. We presented three methods to handle such problems. All suggested methods have been successfully used to prove mutual exclusion of the Peterson algorithm for an arbitrary number of processes [23] and termination of a termination detection algorithm extracted from Ricart and Agrawalas' algorithm [25].

It is left to study the relation ship between these methods. Is one stronger than the other or are they incomparable, and one works on some instances while the second works on other.

In this paper we consider only safety properties. It is important to extend our methods to verify liveness properties as well.

In order to apply regular model checking or any of the methods we have developed here to a given verification problem, the verification problem must be modeled by languages and bi-languages. In several cases, in order to model the system, we used an encoding which is an abstraction of the system. It is interesting to see if one can automatically verify the correctness of such an abstraction, or even more, automatically produce a correct encoding (abstraction).

References

1. P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. *CAV'98*, *LNCS*, volume 1427, p. 305–318. Springer-Verlag, 1998.
2. P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. *CAV'99*, volume 1633 of *LNCS*, p. 134 Springer-Verlag, 1999.
3. K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.
4. J-P. Bodeveix and M. Filali. Experimenting acceleration methods for the validation of infinite state systems. In *DSVV'2000*, Taipei, Taiwan, April 2000.

5. B. Boigelot and P. Godefroid. Symbolic verification of coomunication protocols with infinite state spaces using QDDs. *CAV'96*, volume 1102 of *LNCS*, p. 1–12. Springer-Verlag, 1996.
6. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, LNCS Springer-Verlag, 1997.
7. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. *CONCUR'97*, *LNCS*, volume 1243, p. 135. Springer-Verlag, 1997.
8. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with non-regular sets of configurations. In ICALP'97, volume 1256 of *LNCS* Springer-Verlag, 1997.
9. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touilli. Regular model checking. *CAV'00*, volume 1855 of *LNCS*, p. 403–418. Springer-Verlag, 2000.
10. A. Bouajjani and O. Maler. Reachability analysis of push-down automata. In *Workshop on Infinite-state Systems*, Pisa, 1996.
11. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
12. O. Burkat and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.
13. E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *CONCUR'95*, p. 395–407, Philadelphia, PA, August 1995.
14. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV'98*, LNCS, p. 268–279, 1998.
15. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL'97.
16. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. *CAV'00*, volume 1855 of *LNCS*, p. 232–247. Springer-Verlag, 2000.
17. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, 1997.
18. L. Fribourg and H. Olsen. Reachability sets of parametrized rings as regular languages. In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, volume 9, July 1997.
19. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.
20. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf, editor, *Proceedings of TACAS'00*, LNCS, 2000. To Appear.
21. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *CAV'97*, LNCS , p. 424–435. Springer-Verlag, 1997.
22. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *POPL'97*, Paris, 1997.
23. G.L. Peterson. Myths about the mutual exclusion problem. *Info. Proc. Lett.*, 12(3):115, 1981.
24. A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. *CAV'00*, volume 1855 of *LNCS*, p. 328–343. Springer-Verlag, 2000.

25. G. Ricart and A.K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM*, 24(1):9–17, 1981. Corr. ibid. 1981, p.581.
26. G. Senizergues. The equivalence problem for deterministic pushdown automata is decidable. volume 1256 of *LNCS*, p. 671–681, 1997.
27. E. Shahar. The tlv system and its application. Master's thesis, Weizmann Institute, 1996.
28. Colin Stirling. Decidability of dpda equivalence. *FODDACS'01*, LNCS Springer-Verlag, 2001.
29. P. Wolper and B. Boigelot. Symbolic verification with periodic sets. In D. Dill, editor, *CAV'94*, volume 818 of *LNCS*, p. 55–67. Springer-Verlag, 1994.
30. Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. *CAV'98*, *LNCS*, volume 1427, p. 88–97. Springer-Verlag, 1998.

Relations Between Communication Complexity, Linear Arrangements, and Computational Complexity *

Jürgen Forster[1], Matthias Krause[2], Satyanarayana V. Lokam[3], Rustam Mubarakzjanov[4], Niels Schmitt[1], and Hans Ulrich Simon[1]

[1] Fakultät für Mathematik, Ruhr-Universität Bochum, D-44780 Bochum, Germany
{forster,nschmitt,simon}@lmi.ruhr-uni-bochum.de
[2] Institut für Informatik, Universität Mannheim, D-68131 Mannheim, Germany
krause@th.informatik.uni-mannheim.de
[3] School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540, USA
satya@math.ias.edu
[4] Fakultät für Informatik, Universität Trier, D-54286 Trier, Germany
rustam@TI.uni-trier.de

Abstract. Recently, Forster [7] proved a new lower bound on probabilistic communication complexity in terms of the operator norm of the communication matrix. In this paper, we want to exploit the various relations between communication complexity of distributed Boolean functions, geometric questions related to half space representations of these functions, and the computational complexity of these functions in various restricted models of computation. In order to widen the range of applicability of Forster's bound, we start with the derivation of a generalized lower bound. We present a concrete family of distributed Boolean functions where the generalized bound leads to a linear lower bound on the probabilistic communication complexity (and thus to an exponential lower bound on the number of Euclidean dimensions needed for a successful half space representation), whereas the old bound fails. We move on to a geometric characterization of the well known communication complexity class C-PP in terms of half space representations achieving a large margin. Our characterization hints to a close connection between the bounded error model of probabilistic communication complexity and the area of large margin classification. In the final section of the paper, we describe how our techniques can be used to prove exponential lower bounds on the size of depth-2 threshold circuits (with still some technical restrictions). Similar results can be obtained for read-k-times randomized ordered binary decision diagram and related models.

* This work has been supported in part by the ESPRIT Working Group in Neural and Computational Learning II, NeuroCOLT2, No. 27150. The first, fifth, and sixth author was supported by the Deutsche Forschungsgemeinschaft Grant SI 498/4-1. The third author was partially supported by NSF Grants CCR-9988359 and DMS-9729992.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 171–182, 2001.
© Springer-Verlag Berlin Heidelberg 2001

1 Introduction

Linear algebraic techniques play a pervasive role in the study of computational complexity. In particular, several matrix functions related to rank and its robustness under various changes to the matrix have been extensively used in two-party communication complexity, many models of circuits, decision trees, branching programs, and span programs. Often such functions arise from or are equivalent to nice, natural geometric questions about the matrices. This paper studies certain geometric realizations of real matrices (with no zero entries) and applies results about such realizations of explicit matrices to derive lower bounds in communication complexity, threshold circuits, and ordered binary decision diagrams.

The main mathematical problem we investigate is as follows. Let $M \in \mathbb{R}^{X \times Y}$ be a matrix with non-zero entries. We say M can be *realized* by a k-dimensional linear arrangement if there are vectors $u_x, v_y \in \mathbb{R}^k$ for $x \in X$ and $y \in Y$ such that for all $(x, y) \in X \times Y$, $\operatorname{sign}(M_{x,y}) = \operatorname{sign}(\langle u_x, v_y \rangle)$. The two functions of interest are **i)** the *minimal dimension* $d(M)$ of M, defined to be the minimal k such that M can be realized by a k-dimensional linear arrangement, and **ii)** the *maximal margin*[1] $\mu(M)$ of M, defined to be the maximum over all realizations (of all dimensions) $\{u_x, v_y : x \in X, y \in Y\}$ of $\min\{\frac{|\langle u_x, v_y \rangle|}{L_2(u_x) \cdot L_2(v_y)} : x \in X, y \in Y\}$. Here, L_2 denotes the Euclidean norm.

The minimal dimension can be interpreted in terms of rank. It is easy to see that the minimal dimension of a matrix $M \in \mathbb{R}^{X \times Y}$ is the minimal rank of any real matrix $M' \in \mathbb{R}^{X \times Y}$ such that for all $(x, y) \in X \times Y$, $\operatorname{sign}(M_{x,y}) = \operatorname{sign}(M'_{x,y})$. Note that the minimal dimension depends only on the sign-pattern of M but not on its actual (non-zero) entries. Hence, we can ask the following question: given a sign-pattern (a matrix with entries ± 1), what is the smallest rank of a matrix that obeys this sign-pattern (agrees with the given matrix in sign)? Stated differently, the minimal dimension of a ± 1-matrix describes the robustness of its rank with respect to sign-preserving changes.

We review here, briefly, some history of the problem. Paturi and Simon [14] introduced the model of *unbounded error probabilistic communication complexity*. They showed that the problem of estimating the minimal dimension of matrices with ± 1 entries is essentially equivalent to estimating the unbounded error probabilistic communication complexity of the corresponding Boolean function. Alon, Frankl, and Rödl [1] showed that for *almost all* $n \times n$ matrices with ± 1 entries, the minimal dimension is at least $\Omega(n)$ implying that for most Boolean functions the unbounded error probabilistic communication complexity is asymptotically as large as it can be (linear in the length of the input). However, proving even a superlogarithmic lower bound on the minimal dimension of an explicit matrix remained a difficult open question. Recently, Forster [7] solved this long standing open question. He showed a general lower bound on the minimal dimension of a

[1] A simple argument shows that the number of dimensions can always be reduced to $\min\{|X|, |Y|\}$ without decreasing the margin. A simple compactness argument shows that the maximal margin exists always.

matrix in terms of its operator norm. As a corollary, he derived a lower bound of $\sqrt{n}$ on the minimal dimension of an $n \times n$ Hadamard matrix. This implies a linear lower bound on the unbounded error probabilistic communication complexity of the `inner product mod 2` function. Forster's results [7] also include upper bounds on maximal margins of explicit matrices. The problem of maximal margins is motivated by the excellent empirical performance of maximal margin classifiers [6] (these are learning algorithms that calculate the hyperplane with largest margin on a sample and use that hyperplane to classify new instances). As outlined in [4], the status of the problem for maximal margins was quite similar to the status of the problem for minimal dimensions: although the maximal margin of most matrices $M \in \{-1,+1\}^{X \times Y}$ is (provably) not substantially larger than the trivial margin (which can be shown[2] to be $\max\{|X|^{-1/2}, |Y|^{-1/2}\}$), the question of proving such an upper bound for explicit quadratic matrices has been open. Forster's result gives an upper bound on the maximal margin of a matrix in terms of its operator norm. For the Hadamard matrix, this general bound yields the best possible upper bound on the maximal margin (matching the trivial lower bound).

In this paper, we first generalize Forster's bound [7] on the minimal dimension. The general bound applies to any real matrix with non-zero entries, not just ± 1 matrices. We demonstrate this generality by showing strong bounds for a class of matrices for which Forster's [7] old bound fails.

Next, we give a characterization of the communication complexity class C-PP (analog of the well-known Turing Machine complexity class PP) in terms of maximal margins. We show that C-PP is exactly the class of languages whose communication matrices (with ± 1 entries) have a maximal margin of $2^{-\mathrm{polylog}(n)}$, where n is the length of the inputs given to the two processors. An interesting ingredient of our proof is a random projection technique introduced by Arriaga and Vempala [3] in the context of learning algorithms. A result due to Halstenberg and Reischuk [10] characterizes C-PP in terms of a probabilistic *one-way* protocol that uses at most $2^{\mathrm{polylog}(n)}$ messages and achieves an error bound of $2^{-\mathrm{polylog}(n)}$. Here, we give a characterization of C-PP in terms of a single parameter that is directly related to the communication matrix.

Our third result are lower bounds on depth-2 threshold circuits in terms of minimal dimension. Proving superpolynomial lower bounds for explicit functions when the threshold gates are allowed to use arbitrary weights is an interesting open question. Hajnal et al. [9] show an exponential lower bound for the `inner product mod 2` function when all the weights used in a depth-2 threshold circuit are polynomially bounded. Here, we strengthen their result by showing that the restriction on the weights of the top gate can be removed. We use lower bounds on minimal dimensions of explicit matrices to derive exponential lower bounds for some explicit functions including `inner product mod 2`. In fact, our lower bounds are exponential when the depth-2 circuit has a threshold gate (with

[2] To achieve margin $|X|^{-1/2}$ use the "trivial embedding" $u_x \in \{0,1\}^X$ and $v_y \in \{-1,+1\}^X$ such that $u_x(i) = 1$ iff $i = x$ and $v_y(x) = M_{x,y}$. Margin $|Y|^{-1/2}$ is obtained analogously.

unrestricted weights) at the top and either threshold gates with polynomial weights or gates computing arbitrary symmetric functions at the bottom. Our results also generalize and strengthen the results of Bruck and Smolensky [5], Krause [11], and Krause and Pudlák [12].

Our last result (contained in the full paper for sake of completeness) is a (sort of "easy-to-get") lower bound on the size of randomized ordered binary decision diagrams (randomized OBDD's).[3] Using the lower bound on the minimal dimension of a Hadamard matrix, it is easy to specify an explicit function (containing the `inner product mod 2` as a subfunction) that requires exponential size randomized OBDD's. Using a standard technique[4], it is not hard to get similar exponential lower bounds for even stronger non-uniform models of computation (like read-k-times randomized OBDDs or various restricted versions of randomized oblivious branching programs).

2 Definitions and Notations

A matrix $M \in \mathbb{R}^{X \times Y}$ induces the function $f(x,y) = M_{x,y}$ and vice versa. Since we often consider the situation that a function $f(x,y)$ is computed by two processors (one of them with x as local input, the other one with y as local input), we speak of a "distributed function". We often blur the distinction between matrices and distributed functions. In other words, we interpret a distributed function as matrix (or vice versa) whenever we find it convenient.

Notions and Facts from Linear Algebra. We assume some familiarity with linear algebra and matrix theory. S^{n-1} denotes the $(n-1)$-dimensional unit sphere, i.e., $S^{n-1} := \{u \in \mathbb{R}^n : \ L_2(u) = 1\}$. The *operator norm* of a matrix $M \in \mathbb{R}^{m \times n}$ is defined as follows:

$$\|M\| := \max\{L_2(Mu) : u \in S^{n-1}\} \tag{1}$$

It is well known that for every matrix M the equalities $\|M\| = \|M^\top\|$ and $\|MM^\top\| = \|M\|^2$ hold. If $M \in \mathbb{R}^{X \times X}$ is symmetric, then $\|M\|$ coincides with the largest eigenvalue of M. If $M \in \mathbb{R}^{X \times X}$ is orthogonal, then $\|M\| = 1$. If $M \in \mathbb{R}^{X \times Y}$ has pairwise orthogonal columns, then $\|M\|$ coincides with the Euclidean length of the longest row vector of M.

Probabilistic Communication Complexity. A *two-way* probabilistic communication protocol is a probabilistic algorithm for two processors Π_0 and Π_1 that

[3] An ordered binary decision diagram (OBDD) is a restricted model of a branching program that has been extensively studied both in complexity theory and in applications such as automatic verification. The reader interested in more information about OBDD's is referred to the book of Ingo Wegener [16], where also many pointers to the literature can be found.

[4] first introduced by Alon and Maass [2], later extended by Krause and Waack [13], and further elaborated by different people (see [15], for instance)

computes a distributed function $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$. Both processors have unbounded computational power. Π_0 sees only the first part, x, and Π_1 sees only the last part, y, of the input $(x,y) \in \{0,1\}^n \times \{0,1\}^n$. Obviously there has to be some communication between the two processors to calculate $f(x,y) \in \{0,1\}$. The processors can communicate by exchanging messages $b \in \{0,1\}^*$. The computation takes place in *rounds*. In each round one of the processors is active, in odd rounds it is Π_0 and in even rounds it is Π_1. The active processor probabilistically (depending on the part of the input it knows and on the past messages) chooses a message according to the communication protocol. In the final round the active processor probabilistically chooses the result of the computation. For a *one-way* probabilistic communication protocol there are only two rounds: Π_0 randomly selects a message, sends it to Π_1, and Π_1 randomly performs the binary decision of either accepting the input or rejecting it.

We say that a protocol computes the distributed function $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ with unbounded error if for all inputs $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ the correct output is calculated with probability greater than $\frac{1}{2}$. (Since the slightest edge over random guessing is already sufficient, this model is sometimes called the *unbounded error* model.) The length of a communication protocol is $\lceil \log_2 N \rceil$, where N is the number of distinct message sequences that can occur in computations that follow the protocol. The communication complexity $\mathrm{PComm}(f)$ of a distributed function $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ is the smallest length that any communication protocol that correctly computes f can have.

We briefly note that Paturi and Simon [14] have shown that, in the unbounded error model, one-way and two-way protocols have almost the same power. In fact, any probabilistic two-way protocol of length k can be converted into a one-way protocol with length at most $k+1$. Furthermore, the minimal dimension $d(f)$ of a distributed Boolean function $f(x,y)$ is closely related to its probabilistic communication complexity $\mathrm{PComm}(f)$. Paturi and Simon [14] have proven the following relation:

$$\lceil \log d(f) \rceil \leq \mathrm{PComm}(f) \leq \lceil \log d(f) \rceil + 1 \tag{2}$$

We will show in this paper that the following quantity is related to the margin: given a probabilistic protocol, let $\varepsilon(x,y)$ denote the difference between the probability that (x,y) is accepted and the probability that (x,y) is rejected. Thus $f(x,y) = \mathrm{sign}(\varepsilon(x,y))$ for any probabilistic protocol that correctly computes f. The *error bound* of the protocol is defined as $\min_{x \in X, y \in Y} |\varepsilon(x,y)|$. The class C-PP consists of all functions f that can be correctly computed by a probabilistic communication protocol of length $\mathrm{polylog}(n)$ with error bound $2^{-\mathrm{polylog}(n)}$. Halstenberg and Reischuk [10] have shown that the class C-PP does not change if we allow only one-way protocols. The class C-PP is one of the main issues in the bounded-error model for probabilistic communication complexity.

3 Bounds on the Dimension and on the Margin

We first briefly review some of the main results in [7] and [8]. Thereafter, we present a new lower bound on the minimal dimension that generalizes the old

bound from [7]. The generalized bound can lead to exponential lower bounds in cases where the old bound fails.

Known Bounds. Recently the following lower bound on the minimal dimension was proven:

Theorem 1 (Forster [7]). *For each matrix* $M \in \{-1,+1\}^{X\times Y}$:

$$d(M) \geq \frac{\sqrt{|X|\cdot|Y|}}{\|M\|}.$$

Corollary 1 (Forster [7]). *Let* $M \in \{-1,+1\}^{X\times Y}$ *and* $M' \in \mathbb{R}^{X\times Y}$ *such that* $\operatorname{sign}(M_{x,y}) = \operatorname{sign}(M'_{x,y})$ *for all* $x \in X, y \in Y$. *Then the following holds:*

$$\operatorname{PComm}(M) \geq \log\left(\frac{\sqrt{|X|\cdot|Y|}}{\|M\|}\right),$$

$$\operatorname{rank}(M') \geq \frac{\sqrt{|X|\cdot|Y|}}{\|M\|}.$$

The lower bound on $\operatorname{PComm}(M)$ follows directly from Theorem 1 and (2). The lower bound on $\operatorname{rank}(M')$ follows from the well known fact that each matrix of rank k can be realized by a k-dimensional linear arrangement and vice versa.

As outlined in Forster [7], Theorem 1 allows to present concrete families of distributed Boolean functions whose minimal dimension grows exponentially in n. Thus, their probabilistic communication complexity grows linearly in n. Here is the example given in [7]:

Example 1. Let $\operatorname{ip}_n(x,y) := (-1)^{x^\top y}$, where $x, y \in \mathbb{Z}_2^n$, be the `inner product mod 2` function. It is well known that the corresponding matrix H such that $H_{x,y} = \operatorname{ip}_n(x,y)$ is a Hadamard matrix. H has operator norm $2^{n/2}$ because it has orthogonal columns and each row vector has Euclidean length $2^{n/2}$. According to Theorem 1 and Corollary 1:

$$d(\operatorname{ip}_n) \geq 2^{n/2} \text{ and } \operatorname{PComm}(\operatorname{ip}_n) \geq n/2 \tag{3}$$

Forster, Schmitt, and Simon [8] proved the following upper bound on the maximal margin:

Theorem 2 (Forster, Schmitt, Simon). *For each matrix* $M \in \mathbb{R}^{X\times Y}$ *with no zero entries:*

$$\mu(M) \leq \frac{\sqrt{|X|}\cdot\|M\|}{\sqrt{\sum_{y\in Y}\left(\sum_{x\in X}|M_{x,y}|\right)^2}}\ .$$

Note that Theorem 2 implies the following upper bound on the maximal margin of the inner product mod 2 function:[5]

$$\mu(\operatorname{ip}_n) \leq 2^{-n/2}. \tag{4}$$

[5] This upper bound had been shown already by Forster [7]. He derived it from a theorem that is slightly weaker than Theorem 2.

A Generalized Lower Bound on the Minimal Dimension. Theorem 1 allowed us to derive an exponential lower bound on $d(\mathrm{ip}_n)$ because the matrix induced by ip_n, the Hadamard matrix, has 2^n orthogonal columns. However, Theorem 1 may also fail to certify valid exponential lower bounds. For this reason, we present a generalized lower bound which brings more functions in the reach of our machinery:

Theorem 3. *For each matrix $M \in \mathbb{R}^{X \times Y}$ with no zero entries:*

$$d(M) \geq \frac{\sqrt{|X|\,|Y|}}{\|M\|} \min_{x \in X, y \in Y} |M_{x,y}| \ .$$

Proof. Assume that there are vectors $u_x, v_y \in \mathbb{R}^k$ such that $\mathrm{sign}(M_{x,y}) = \mathrm{sign}(\langle u_x, v_y \rangle)$ for all $x \in X$, $y \in Y$. Obviously we can assume that the minimum $\min_{x \in X, y \in Y} |M_{x,y}|$ is 1. It has been shown by Forster [7] that a given linear arrangement u_x, v_y can be modified such that $u_x, v_y \in S^{k-1}$ and $\sum_{x \in X} u_x u_x^\top = \frac{|X|}{k} I_k$. Now we have for all $y \in Y$ that

$$\sum_{x \in X} \underbrace{M_{x,y} \langle u_x, v_y \rangle}_{\geq 0} \overset{|M_{x,y}| \geq 1 \geq |\langle u_x, v_y \rangle|}{\geq} \sum_{x \in X} \langle u_x, v_y \rangle^2 = v_y^\top \left(\sum_{x \in X} u_x u_x^\top \right) v_y = \frac{|X|}{k} \ . \tag{5}$$

It is not hard to show[6] that

$$|Y| \left(\frac{|X|}{k} \right)^2 \leq \sum_{y \in Y} \left(\sum_{x \in X} M_{x,y} \langle u_x, v_y \rangle \right)^2 \leq |X|\,\|M\|^2 \ .$$

Thus $k \geq \frac{\sqrt{|X|\,|Y|}}{\|M\|}$. □

4 An Application of the Generalized Lower Bound

We would like to demonstrate (by means of an example) the greater flexibility provided by Theorem 3. Let $\mathbb{F}_p$ be the prime field of characteristic p, let $\mathbb{F}_p^n$ be the n-dimensional vector space over $\mathbb{F}_p$, and let $\mathbb{P}_{n-1}(\mathbb{F}_p)$ be the $(n-1)$-dimensional projective space. Remember that the elements (projective points) in $\mathbb{P}_{n-1}(\mathbb{F}_p)$ are the 1-dimensional linear subspaces of $\mathbb{F}_p^n$. We assume that the projective points are given by their homogeneous coordinates. We say that a projective point $Q = (q_1, \ldots, q_n)$ is *orthogonal* to a projective point $Q' = (q'_1, \ldots, q'_n)$, denoted as $Q \perp Q'$, if $\sum_{i=1}^n q_i q'_i = 0$. The Boolean function $\mathrm{ORT}_{p,n} : \mathbb{P}_{n-1}(\mathbb{F}_p) \times \mathbb{P}_{n-1}(\mathbb{F}_p) \to \{-1, 1\}$

$$\mathrm{ORT}_{p,n}(Q, Q') := \begin{cases} 1, & \text{if } Q \perp Q' \\ -1, & \text{otherwise} \end{cases} \tag{6}$$

[6] The first inequality follows directly from (5). The second inequality follows from some algebraic manipulations that are found in the full paper.

is the indicator function of the orthogonality relation. In the full paper, we show that the operator norm of $\mathrm{ORT}_{p,n}$ is too small for certifying an exponential lower bound on the minimal dimension with Theorem 1. In contrast to this, the following can be shown by means of Theorem 3 and Theorem 2 (and by means of a nice trick from [13]):

Theorem 4. *Asymptotically, the minimal dimension and the maximal margin of* $\mathrm{ORT}_{p,n}$ *are bounded as follows:*

$$d(\mathrm{ORT}_{p,n}) \geq p^{n/2-1}(1-o(1))$$

$$\frac{1}{p^{(n-1)/2}}(1-o(1)) \leq \mu(\mathrm{ORT}_{p,n}) \leq \frac{\sqrt{p}}{2p^{(n-1)/2}}(1+o(1))$$

Proof. Our proof strategy (borrowed from [13]) is to define a function (matrix) $\mathrm{ORT}'_{p,n}$ such that $\mathrm{ORT}_{p,n}$ and $\mathrm{ORT}'_{p,n}$ satisfy the requirements of Theorem 3, and such that the columns of $\mathrm{ORT}'_{p,n}$ are orthogonal (which makes $\|\mathrm{ORT}'_{p,n}\|$ small and, therefore, the lower bound on $d(\mathrm{ORT}_{p,n})$ large). Details follow.

For

$$w_+ := p-1 \text{ and } w_- := \frac{1+p^{(n-2)/2}}{p^{(n-2)/2}},$$

$\mathrm{ORT}'_{p,n}$ is given by

$$\mathrm{ORT}'_{p,n}(Q,Q') := \begin{cases} w_+, & \text{if } Q \perp Q' \\ -w_-, & \text{otherwise} \end{cases}.$$

Obviously, $|\mathrm{ORT}'_{p,n}(Q,Q')| \geq 1$ and $\mathrm{sign}(\mathrm{ORT}'_{p,n}(Q,Q')) = \mathrm{sign}(\mathrm{ORT}_{p,n}(Q,Q'))$ for all Q, Q'. We may therefore apply Theorem 3 and Theorem 2 to $\mathrm{ORT}'_{p,n}$ (instead of $\mathrm{ORT}_{p,n}$). It has been shown by Krause and Waack [13] that $\mathrm{ORT}'_{p,n}$ (viewed as matrix) has orthogonal columns. Its operator norm coincides therefore with the Euclidean length of one of its row vectors (all of which have the same length). The following facts are well known. $N := |\mathbb{P}_{n-1}(\mathbb{F}_p)| = (p^n-1)/(p-1)$. Furthermore, each projective point is orthogonal to $N_+ = (p^{n-1}-1)/(p-1)$ projective points and non-orthogonal to the remaining $N_- = N - N_+ = p^{n-1}$ projective points. We conclude that

$$\|\mathrm{ORT}'_{p,n}\| = \sqrt{N_+ w_+^2 + N_- w_-^2} = p^{n/2}+1.$$

According to Theorem 3: $\quad d(\mathrm{ORT}_{p,n}) \geq \dfrac{\sqrt{N \cdot N}}{\|\mathrm{ORT}'_{p,n}\|} = \dfrac{p^{n/2}-1}{p-1}.$

According to Theorem 2 and by straightforward computation,

$$\mu(\mathrm{ORT}_{p,n}) \leq \frac{\|\mathrm{ORT}'_{p,n}\| \cdot \sqrt{N}}{\sqrt{\sum_Q \left(\sum_{Q'} |\mathrm{ORT}'_{p,n}(Q,Q')|\right)^2}} = \frac{p^{n/2}+1}{2p^{n-1}+p^{n/2}-1}.$$

The trivial margin (obtained from the trivial embedding that was briefly described in the introduction) is $N^{-1/2} = ((p-1)/(p^n-1))^{1/2}$. □

5 Large Margins and Bounded Error

Recall that a family (f_n) of Boolean functions $f_n : \{0,1\}^n \times \{0,1\}^n \to \{-1,+1\}$ belongs to C-PP if there exists a probabilistic one-way protocol that transmits at most polylog(n) bits (uses at most $2^{\text{polylog}(n)}$ messages) and achieves error bound $2^{-\text{polylog}(n)}$.

The goal of this section is to express membership in C-PP in terms of only one parameter: the maximal margin. Here is the main result:

Theorem 5. $(f_n) \in$ C-PP $\Leftrightarrow \mu(f_n) \geq 2^{-\text{polylog}(n)}$.

Theorem 5 follows directly from Lemmas 1, 2, and 3 below. Lemma 2 makes use of the random projection technique that was introduced by Arriaga and Vempala [3]. Lemmas 1 and 3 are implicitly proven in the paper of Paturi and Simon [14] on probabilistic communication complexity with unbounded error. Since they do not explicitly keep track of error bounds (not to speak of margins), a brief sketch of these proofs is given in the full paper.

Lemma 1. *Each probabilistic one-way protocol for f that uses at most N messages and achieves error-bound ε can be converted into an N-dimensional linear arrangement that realizes f with margin $\mu \geq \varepsilon/\sqrt{N}$.*

Lemma 2. *Each linear arrangement (of arbitrarily high dimension) that realizes f with margin μ can be converted into an $\lceil (12(n+1)/\mu)^2 \rceil$-dimensional linear arrangement that realizes f with margin $\mu/2$.*

Proof. The following result (whose proof can be found in [4]), is based on the technique of random projections (from Arriaga and Vempala [3]):
Let $w, x \in \mathbb{R}^r$ be arbitrary but fixed. Let $R = (R_{i,j})$ be a random $(k \times r)$-matrix such that the entries $R_{i,j}$ are i.i.d. according to the normal distribution $N(0,1)$. Consider the random projection $u_R := \frac{1}{\sqrt{k}}(Ru) \in \mathbb{R}^k$ for all $u \in \mathbb{R}^k$. Then the following holds for every constant $\mu > 0$:

$$\Pr_R \left[|\langle w_R, x_R \rangle - \langle w, x \rangle| \geq \frac{\mu}{2} \left(L_2(w)^2 + L_2(x)^2 \right) \right] \leq 4e^{-\mu^2 k/8}.$$

This result can be used in the obvious way to guarantee the existence of a random projection that maps an r-dimensional linear arrangement that realizes f with margin μ to a k-dimensional linear arrangement that realizes f with margin $\mu/2$. The dimension k must exceed a critical threshold that depends on μ and n (but does not depend on r). If the computations are carried out carefully, it turns out that $k := \lceil (12(n+1)/\mu)^2 \rceil$ is sufficiently large. We omit the (somewhat tedious) computations in this abstract. □

Lemma 3. *Each N-dimensional linear arrangement that realizes f with margin μ can be converted into a probabilistic one-way protocol that uses at most $2N$ messages and achieves error bound $\varepsilon \geq \mu/\sqrt{N}$.*

6 Minimal Dimension and Computational Complexity

In this section, we show how lower bounds on the minimal dimension can be used to derive improved lower bounds on the complexity of Boolean functions in various non-uniform models of computation. In this brief abstract, we focus on depth-2 threshold circuits. The reader interested in exponential lower bounds on the size of randomized OBDD's and related models is referred to the full paper.

The main result in this section states that (loosely speaking) Boolean functions with "high" minimal dimension cannot be computed by "small" (somewhat technically restricted) depth-2 threshold circuits. Part a) of this theorem strengthens the lower bound of Hajnal et al. [9] and Part b) generalizes and strengthens the results of Bruck and Smolensky [5], Krause [11], and Krause and Pudlák [12]. Note that for technical reasons we assume that the top threshold gate is ± 1-valued. The threshold gates on the bottom level are $\{0,1\}$-valued.

Theorem 6. *Let (f_n) be a family of distributed Boolean functions $f_n : \{0,1\}^n \times \{0,1\}^n \to \{-1,+1\}$. Suppose (f_n) is computed by depth-2 threshold circuits in which the top gate is a linear threshold gate (with* unrestricted *weights). Then the following holds:*

a) *If the bottom level has s linear threshold gates using integer weights of absolute value at most W, then $s = \Omega\left(\frac{d(f_n)}{nW}\right)$. In particular, $s = \Omega\left(\frac{2^{n/2}}{nW}\right)$ for $f_n = \mathrm{ip}_n$, and $s = \Omega\left(\frac{p^{n/2-1}}{nW}\right)$ for $f_{n\lceil \log p\rceil} = \mathrm{ORT}_{p,n}$.*

b) *If the bottom level has s gates computing symmetric functions, then $s = \Omega\left(\frac{d(f_n)}{n}\right)$. In particular, $s = \Omega\left(\frac{2^{n/2}}{n}\right)$ for $f_n = \mathrm{ip}_n$, and $s = \Omega\left(\frac{p^{n/2-1}}{n}\right)$ for $f_{n\lceil \log p\rceil} = \mathrm{ORT}_{p,n}$.*

Note that the specific bounds for ip_n and $\mathrm{ORT}_{p,n}$ follow immediately from the general bound for f_n by applying (3) and Theorem 4.

The proof of this theorem is based on Theorem 1 and on ideas from the results cited above. We start with two lemmas.

Lemma 4. *Let $G : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$ be a threshold function where for $x, y \in \{0,1\}^n$, $G(x,y) = 1$ iff $\sum_{i=1}^n \alpha_i x_i + \sum_{i=1}^n \beta_i y_i \geq \mu$ for weights $\alpha_i, \beta_i, \mu \in \mathbb{Z}$. Then, G (viewed as a matrix) has rank at most $\min\{\sum_{i=1}^n |\alpha_i|, \sum_{i=1}^n |\beta_i|\}+1$.*

Proof. W.l.o.g. let $\sum_{i=1}^n |\alpha_i| \leq \sum_{i=1}^n |\beta_i|$. Let $\alpha_{\min}$ ($\alpha_{\max}$) be the minimal (maximal) value taken by $\sum_{i=1}^n \alpha_i x_i$ as x ranges over all possible inputs in $\{0,1\}^n$. As the weights are integers, this sum takes at most $\alpha_{\max} - \alpha_{\min} + 1$ distinct values. We partition the rows of G according to the weight contributed by x and, within each block of these rows, we partition the columns into two groups depending on whether or not the weight contributed by y together with that of any row of that block exceeds the threshold μ or not. Specifically, define the following sets

of entries of G for all α such that $\alpha_{\min} \leq \alpha \leq \alpha_{\max}$:

$$S_{\alpha,0} := \{(x,y) : \sum_{i=1}^{n} \alpha_i x_i = \alpha \text{ and } \sum_{i=1}^{n} \beta_i y_i < \mu - \alpha\},$$
$$S_{\alpha,1} := \{(x,y) : \sum_{i=1}^{n} \alpha_i x_i = \alpha \text{ and } \sum_{i=1}^{n} \beta_i y_i \geq \mu - \alpha\}.$$

Let $G_{\alpha,0}$ and $G_{\alpha,1}$ be (disjoint) submatrices of G defined by the entries $S_{\alpha,0}$ and $S_{\alpha,1}$ respectively. It is clear that $G_{\alpha,0}$ is an all-0 matrix and $G_{\alpha,1}$ is an all-1 matrix for any α. Furthermore $G = \sum_{\alpha}(G_{\alpha,0} + G_{\alpha,1})$. Hence, by subadditivity of rank we see that the rank of G is at most the number of distinct values taken by α. The latter is bounded above by $\alpha_{\max} - \alpha_{\min} + 1 \leq \sum_{i=1}^{n} |\alpha_i| + 1$. □

Note that the same proof goes through even if we generalize the definition of G by setting $G(x,y) = 1$ iff $\sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n} \beta_i y_i \in T$ for an *arbitrary* subset T of $\mathbb{Z}$. Specifically, we have the following corollary of the proof:

Corollary 2. *Let $G : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$ be a symmetric function in the sense that its value depends only on $\sum_{i=1}^{n}(x_i + y_i)$. Then G (viewed as a matrix) has rank at most $n+1$.*

Lemma 5. *Let $f : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{-1,+1\}$ be a Boolean function computed by a depth-2 threshold circuit C with the top gate using* unrestricted weights *and each of the bottom gates using weights of absolute value at most W. Then there is a real matrix F such that* $\text{sign}(F(x,y)) = f(x,y)$ *for all $x, y \in \{0,1\}^n$ and* $\text{rank}(F) = O(snW)$*, where s is the number of bottom gates.*

Proof. Let the top gate of C have weights $\phi_1, \ldots \phi_s$ and threshold ϕ_0. Hence we can write $f(x,y) = \text{sign}(\sum_{i=1}^{s} \phi_i G_i(x,y) - \phi_0)$, where G_i are the functions computed by the bottom gates. Define the matrix $F := \phi_1 G_1 + \cdots + \phi_s G_s - \phi_0 J$, where J is the all 1's $2^n \times 2^n$ matrix. It is clear that $f(x,y) = \text{sign}(F(x,y))$ for all $x, y \in \{0,1\}^n$. Moreover, $\text{rank}(F) \leq 1 + \sum_{i=1}^{s} \text{rank}(G_i) \leq 1 + s(1 + nW)$ using Lemma 4. □

Using Corollary 2, one can similarly prove that if f is computed by a depth-2 circuit with a threshold gate at the top and s symmetric gates at the bottom, then there is a matrix F of rank $O(sn)$ that sign-represents f.

Proof (of Theorem 6). By Lemma 5, if a depth-2 threshold circuit computes $f_n(x,y)$, then there is a matrix F_n such that $\text{sign}(F_n(x,y)) = \text{sign}(f_n(x,y))$ and $\text{rank}(F_n) = O(snW)$. On the other hand, $\text{rank}(F_n) \geq d(f_n)$ (because there is always a $\text{rank}(F_n)$-dimensional linear arrangement for F_n, which is then also a linear arrangement for f_n). Comparing the upper and lower bounds on $\text{rank}(F)$, we get $snW = \Omega(d(f_n))$. This proves part a) of the theorem. Part b) is proved similarly by means of Corollary 2. □

References

1. N. Alon, P. Frankl, and V. Rödl. Geometrical realization of set systems and probabilistic communication complexity. In *Proceedings of the 26th Symposium on Foundations of Computer Science*, pages 277–280, 1985.
2. N. Alon and W. Maass. Meanders and their applications in lower bound arguments. *Journal of Computer and System Sciences*, 37:118–129, 1988.
3. Rosa I. Arriaga and Santosh Vempala. An algorithmic theory of learning: Robust concepts and random projection. In *Proceedings of the 40'th Annual Symposium on the Foundations of Computer Science*, pages 616–623, 1999.
4. Shai Ben-David, Nadav Eiron, and Hans Ulrich Simon. Limitations of learning via embeddings in euclidean half-spaces. In *Proceedings of the 14th Annual Workshop on Computational Learning Theory*, pages 385–401. Springer Verlag, 2001.
5. Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions and spectral norms. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, pages 632–641, 1990.
6. Nello Christianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
7. Jürgen Forster. A linear bound on the unbounded error probabilistic communication complexity. In *Proceedings of the 16th Annual Conference on Computational Complexity*, pages 100–106, 2001.
8. Jürgen Forster, Niels Schmitt, and Hans Ulrich Simon. Estimating the optimal margins of embeddings in euclidean half spaces. In *Proceedings of the 14th Annual Workshop on Computational Learning Theory*, pages 402–415. Springer Verlag, 2001.
9. A. Hajnal, W. Maass, P. Pudlàk, M. Szegedy, and G. Turán. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46:129–154, 1993.
10. Bernd Halstenberg and Rüdiger Reischuk. Different modes of communication. *SIAM Journal on Computing*, 22(5):913–934, 1993.
11. Matthias Krause. Geometric arguments yield better bounds for threshold circuits and distributed computing. *Theoretical Computer Science*, 156:99–117, 1996.
12. Matthias Krause and Pavel Pudlák. On the computational power of depth-2 circuits with threshold and modulo gates. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 48–57, 1993.
13. Matthias Krause and Stephan Waack. Variation ranks of communication matrices and lower bounds for depth two circuits having symmetric gates with unbounded fan-in. In *Proceedings of the 32nd Symposium on Foundations of Computer Science*, pages 777–782, 1991.
14. Ramamohan Paturi and Janos Simon. Probabilistic communication complexity. *Journal of Computer and System Sciences*, 33(1):106–123, 1986.
15. Martin Sauerhoff. On the size of randomized OBDDs and read-once branching programs for k-stable functions. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 488–499. Springer Verlag, 1999.
16. Ingo Wegner. *Branching programs and binary decision diagrams — theory and applications*. Monographs on Discrete Mathematics and Applications. SIAM, 2001.

Optimal, Output-Sensitive Algorithms for Constructing Upper Envelope of Line Segments in Parallel

Neelima Gupta, Sumit Chopra, and Sandeep Sen

[1] Department of Computer Science, Hansraj College, Delhi University, New Delhi 110007, India. neelima_research@yahoo.com
[2] Department of Computer Science, Hansraj College, Delhi University, New Delhi 110007, India. spchopra@mantraonline.com
[3] Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India. ssen@cse.iitd.ernet.in

Abstract. In this paper we focus on the problem of designing very fast parallel algorithms for constructing the upper envelope of straight-line segments that achieve the $O(n \log H)$ work-bound for input size n and output size H. Our algorithms are designed for the arbitrary CRCW PRAM model. We first describe an $O(\log n \cdot (\log H + \log \log n))$ time deterministic algorithm for the problem, that achieves $O(n \log H)$ work bound for $H = \Omega(\log n)$. We present a fast randomized algorithm that runs in expected time $O(\log H \cdot \log \log n)$ with high probability and does $O(n \log H)$ work. For $\log H = \Omega(\log \log n)$, we can achieve the running time of $O(\log H)$ while simultaneously keeping the work optimal. We also present a fast randomized algorithm that runs in $\tilde{O}(\log n / \log k)$ time with nk processors, $k > \log^{\Omega(1)} n$. The algorithms do not assume any input distribution and the running times hold with high probability.

1 Introduction

The upper envelope of a set of n line segments in the plane is an important concept in visibility and motion planning problems. The segments are regarded as opaque obstacles, and their upper envelope consists of the portion of the segments visible from the point $(0, +\infty)$. The complexity of the upper envelope is the number of distinct pieces of segments that appear on it. If the segments are nonintersecting, then the complexity of their upper envelope is linear in n. On the other hand, if the segments are allowed to intersect, then the worst case complexity of the upper envelope increases to $O(n\alpha(n))$, where $\alpha(n)$ is the functional inverse of Ackermann's function [2].

There exists an $O(n \log n)$ time algorithm to compute the upper envelope of n line segments, and this is worst case optimal [15]. However, this is true only if the output size, i.e., the number of vertices (or the edges) of the upper envelope, is large. More specifically, the time-bound of $O(n \log n)$ is tight when the ordered output size is $\Omega(n)$. However if the output size is small then we should be able to do much better. The output-size of a problem is an important parameter in

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 183–194, 2001.
© Springer-Verlag Berlin Heidelberg 2001

measuring the efficiency of an algorithm and one can get considerably superior algorithms in terms of it. There exists an $O(n \log H)$ algorithm for the problem [17], where H is the output size, which implies a linear time algorithm for constant output size.

We are aiming at designing an output size sensitive parallel algorithm that speeds up optimally with output size in the sub-logarithmic time domain.

In the context of sequential algorithms, it has been observed that the upper envelope of n line segments can be computed in $O(n\alpha(n) \log n)$ time, by a straight forward application of divide-and-conquer technique. Hershberger describes an optimal $O(n \log n)$ algorithm by reorganizing the divide-and-conquer computation [15].

Clarkson describes a randomized $O(n\alpha(n) \log n)$ algorithm for computing a face in an arrangement of line segments [8]. The upper envelope of line segments can be viewed as one such face. Guibas et al. [11] , gave a deterministic algorithm which computed a face for a collection of line segments in $O(n\alpha^2(n) \log n)$ time.

The process of gift wrapping amounts to locating an extreme segment of the upper envelope and then "walking" along successive segments of the envelope which are defined by the points of intersection of the lines in the given arrangement. This results in a simple $O(nH)$ time algorithm. Franck Nielsen and Mariette Yvinec presents a deterministic sequential output-sensitive algorithm that computes the upper envelope in $O(n \log H)$ time [17]. Their algorithm is based on the marriage-before-conquest paradigm to compute the convex hull of a set of fixed planar convex objects, which they also apply to compute the upper envelope of line segments. They use the partitioning technique of Hershberger to get an $O(n \log H)$ time for upper envelope. They claim that their algorithms are easily parallelizable onto EREW PRAM, following the algorithm of S. Akl [3]. This implies a parallel output-sensitive algorithm which is optimal for number of processors bounded by $O(n^z)$, $0 < z < 1$.

Recently Wei Chen and Koichi Wada gave a deterministic algorithm that computes the upper envelope of line segments in $O(\log n)$ time using $O(n)$ processors [7]. If the line segments are nonintersecting and sorted, the envelope can be found in $O(\log n)$ time using $O(n/\log n)$ processors. Their methods also imply a fast sequential result: the upper envelope of n sorted line segments can be found in $O(n \log \log n)$ time.

We present algorithms whose running times are output-sensitive even in the sub-logarithmic time range while keeping the work optimal. For designing fast output-sensitive algorithms, we have to cope with the problem that the output-size is an unknown parameter. Moreover, we also have to rapidly eliminate input line segments that do not contribute to the final output without incurring a high cost - see Gupta and Sen [13] for a more detailed discussion.

We present one deterministic and two randomized algorithms that construct the upper envelope of n line segments. Both our randomized algorithms are of

Las Vegas type, that is we always provide a correct output and the bounds hold with high probability[1].

We first describe a deterministic algorithm for the problem that takes $O(\log n \cdot (\log H + \log\log n))$ time using $O(n/\log n)$ processors. The algorithm is based on the marriage before conquest approach and we use the ideas presented by Neilsen and Yvinec [17] to bound the size of the sub-problems. Our algorithm achieves $O(n \log H)$ work bound for $H = \Omega(\log n)$.

Next we present fast randomized algorithm. The expected running times hold with high probability. The algorithm runs in $O(\log H)$ expected time using n processors for $H > \log^{\varepsilon} n$, $\varepsilon > 0$. For smaller output sizes the algorithm has an expected running time of $O(\log H \cdot \log\log n)$ keeping the number of operations optimal. Therefore for small output sizes our algorithm runs very fast. The algorithm uses the iterative method of Gupta and Sen [13].

We also describe a randomized algorithm that solves the problem in $O(\log n/\log k)$ time with high probability using nk processors for $k > \log^{\Omega(1)} n$. This algorithm is based on the general technique given by Sen to develop sub-logarithmic algorithms [21].

2 Deterministic Algorithm for Upper Envelope

Our algorithm is based on the Marriage-before-conquest technique and it uses the ideas presented by Neilsen and Yvinec [17] to bound the size of the sub-problems.

Let S be the set of n line segments. Let its upper envelope be denoted by $\mathrm{UE}(S)$. Then, if we know that there exists a partition $\cup_{i=1}^{k} P_{\mathrm{i}}$ of S into k subsets such that each subset P_{i}, for $i \in [1, k]$ is a set of non-overlapping line segments, then in the marriage-before-conquest procedure we can bound the size of the sub-problems by $(n/2 + k)$. Thus we transform the set S into a set T of line segments partitioned into subsets, each subset consisting of a set of non-overlapping line segments, such that $\mathrm{UE}(S) = \mathrm{UE}(T)$. We now apply the marriage-before-conquest procedure on the set T. To compute the set T, we partition the set S using a partition tree and use the communication of J. Hershberger to make the size of T linear. With a good estimate of k, we run the marriage-before-conquest procedure on the set T (the number of stages of this procedure depends on k), and reduce the total size of the problem to $n/\log n$, after which the problem is solved directly.

2.1 Transforming the Set S of Line Segments into the Set T

The Vertical Decomposition. Let us call two line segments to be x-separated if there exists a vertical line such that the two line segments lie entirely on the different sides of the vertical line. The vertical decomposition will give rise to a set of x-separated line segments.

[1] The term high probability implies probability exceeding 1-1/n^c for any predetermined constant c where n is the input size.

Let $Q = \{l_1, l_2, \ldots, l_m\}$ be a subset of S. Through each vertex of the upper envelope of Q, we draw a line parallel to y-axis. These parallel lines induce a decomposition of the line segments of Q into smaller line segments called the tiny line segments. We only keep the tiny line segments that participate in the upper envelope. Note that the part of a line segment not contributing to such an upper envelope will not contribute to the final upper envelope. The tiny line segments are x-separated i.e. they are non-overlapping. The number of these tiny line segments is equal to the complexity of the upper envelope. Each tiny line segment is defined by a single line segment.

The Partition Tree. To generate the set T from the set S, we group the line segments of the set S into k groups and compute the vertical decomposition of each group. Thus the tiny line segments formed in this decomposition form the set T, which is partitioned into k subsets, each consisting of non-overlapping segments. However such a grouping technique does not guarantee that the size of the set T i.e. the total number of such tiny line segments, is linear.

We use the technique of J. Hershberger to group the line segments of S. The idea is to create groups so that the size of the vertical decomposition of each group remains linear. Let p be a given integer. We first construct the interval tree of depth $\log p$ on the endpoints of the line segments, as presented by Neilsen and Yvinec [17]. Next we load this tree with the line segments, by allocating n line segments to the nodes of this tree, according to the lowest common ancestor of their end points. By the communication of J. Hershberger [15], the upper envelope of the line segments belonging to an internal node is linear in the number of line segments (He proves this from the fact that line segments belonging to a node cross a vertical line) and the upper envelope of the line segments belonging to different nodes at the same internal level is also linear in the number of line segments since they are x-seperated. We group the line segments of each internal level of the interval tree into groups of size p forming $O(n/p + \log p)$ groups. Also, two line segments belonging to different leaves of the tree are x separated. We group the line segments belonging to the leaves by picking one segment from each leaf and forming a group, obtaining additional n/p groups each of size p. Compute the vertical decomposition of the upper envelope of each group, obtaining $O(2n/p+\log p)$ groups each consisting of $O(p)$ x-seperated tiny line segments. The set T is the union of all these tiny line segments.

Thus we obtain a new set T, which is partitioned into $O(2n/p+\log p)$ subsets, each consisting of p x-seperated tiny line segments, such that $\mathrm{UE}(T) = \mathrm{UE}(S)$. We choose p such that $p \log p < n$.

Lemma 1. *For a fixed integer p, the set T partitioned into $O(2n/p + \log p)$ subsets each of p x-seperated line segments from the set S, such that UE(S) = UE(T) can be generated in $O(\log n \cdot \log p)$ time using $O(n/\log n)$ processors.*

Proof. Computing T requires computing the median, the lowest common ancestor of the endpoints, and $O(2n/p + \log p)$ upper envelopes each of size $O(p)$. All these can be done in the required bounds. □

Remark 1. In fact the same procedure runs in expected $O(\log p)$ time using n processors with high probability.

2.2 The Marriage-Before-Conquest Procedure

We assume that the partition $\cup_{i=1}^{k} P_{\mathrm{i}}$ of the set T into k subsets such that each subset P_{i}, for $i \in [1, k]$ is a set of non-overlapping line segments, is given. Call the two vertical lines passing through the end points of a line segment as the *walls*. Let W be the set of walls and denote by $|W| = 2n$ its cardinality. We define a *slab* as the portion of the Euclidean plane E^2 between two walls. The upper envelope $\mathrm{UE}(S)$ can be described as the x-ordered sequence of edges. The Marriage-before-conquest (*MBC*) procedure computes a subsequence of the edges of $\mathrm{UE}(S)$ included in the slab B formed by the extreme right and extreme left walls. We say that a line segment spans a slab, if it intersects the slab but does not have a wall inside the slab.

Find the median w_{m} of the walls W. Compute the edge b of the upper envelope intersecting w_{m}. Split W into two subsets W_1 (Resp. W_2) consisting of walls that are to the left (Resp. right) of the edge b and not intersecting it. Call the slab formed by W_1 (Resp. W_2) as B_1 (Resp. B_2). The line segments to be considered in a slab are those that have a wall inside the slab and those that span the slab. If a slab has no walls then stop, retain the line segments spanning such a slab (we will take care of these line segments later), else recurse.

Observation 1. At every stage, each sub-problem has at least one output edge.

The Analysis. Our partitioning scheme guarantees that each sub-problem has at most one spanning line segment from a partition (since the line segments belonging to a partition are x-separated). Thus for each sub-problem, there can be at most k spanning line segments. Thus the size of the sub-problems is bounded by $O(n/2 + k)$, where $k = O(2n/p + \log p) = O(n/p)$ for $p \log p < n$ from Lemma 1.

Lemma 2. *From observation 1, if we choose the integer p such that $H \log n \leq p$ the size of the problem reduces to $n/\log n$ after $O(\log p)$ stages of the MBC procedure.*

2.3 The Main Algorithm

The algorithm works in two phases: *The estimation phase.* During this phase we find a good estimate of p ($p < (H \log n)^2$ and $p \log p < n$). We start with a constant value of p, run *MBC* and see if the size of the problem reduces to $n/\log n$ after $\log p$ stages. If this happens for small p our job is done. However otherwise, from Lemma 2 we know that once p is big enough that is $p \geq H \log n$, we are guarenteed this to happen. Thus p is always $< (H \log n)^2$. For every p we also check if $p \log p > n$ ($n < p \log p$, $p < (H \log n)^2, \Rightarrow H > n^{\varepsilon}$). If so, then use an $O(n, \log n)$ algorithm to solve the problem directly and stop. *The terminating phase.* During this phase we solve the problem (of size $n/\log n$) directly.

The Analysis. For a fixed p from Lemma 1, the set T can be generated in $O(\log n \cdot \log p)$ time using $O(n/\log n)$ processors. The total time spent during the *MBC* procedure is $O(\log n \cdot \log p)$ using $O(n/\log n)$ processors.

Let p_e denote the estimate of p, then $p_e < (H \log n)^2$. If p_i is the i^{th} estimation of p, then the total time spent in this phase is $< O(\log n \sum \log p_i) = O(\log n \cdot \log p_e) = O(\log n \cdot (\log H + \log\log n))$. The problem of size $n/\log n$ can be solved in $O(\log n)$ time. Thus we have the following theorem.

Theorem 1. *The upper envelope of n line segments can be constructed in $O(\log n \cdot (\log H + \log\log n))$ time using $O(n/\log n)$ processors in a deterministic CRCW PRAM.*

3 The Randomized Optimal Algorithm

In this section we present a randomized algorithm using $O(n)$ processors. The algorithm is an iterative one rather than recursive. The underlying algorithm is similar to the algorithm described for the planar convex hulls by Gupta and Sen [13]. Hence the entire analysis of their algorithm goes through in our case also. However here we detail out the steps that are specific to our problem. We assume for simplicity that no more than two line segments intersect at the same point and the abscissa and ordinate of the end points of no two line segments and that of points of intersection are the same. To take care of the gaps we also introduce an extra line segment L lying below all the line segments.

The idea is to construct the upper envelope of the random sample R of the set S of staright line segments and filter out the redundant segments that do not contribute to the upper envelope of S. We pickup a sample of line segments of constant size, compute its upper envelope, discard all the redundant line segments and iterate on the reduced problem. In each successive iteration we square the size of the sample to be chosen. We iterate until either the size of the problem reduces to some threshold or the sample size becomes greater than some threshold. At this point, if the problem size reduces to less than n^{ε}, we use a brute force algorithm to solve the problem in constant time, else we use the algorithm described by Chen and Wada [7] to compute the upper envelope directly.

To prove any interesting results we must determine how quickly the problem size decreases. The Random Sampling Lemmas discussed in the next section guarantee that when a sample of size $\Omega(H^2)$ is chosen, the problem size reduces fast.

3.1 The Random Sampling Lemmas

For any subset $P \subseteq S$ consider the the two adjacent edges of the upper envelope of P. Draw two vertical lines, one through the left end point of the left edge and the other through the right end point of the right edge. We define a slab as the portion of the Euclidian plane E^2 between these two vertical lines. A configuration σ (which we call region) is the region lying above the upper envelope in

such a slab. We say that the line segments adjacent to σ define σ. Notice that we include the line segment L (defined previously) in every subset P of S. From now onwards, whenever we talk of a subset P of S we would actually be meaning $P \cup L$.

We define a configuration space $\Pi(S)$ as the set of all such configurations defined by all the subsets of S. The set of line segments intersecting a region σ is called the conflict list of σ and is denoted by $L(\sigma)$ and its cardinality $|L(\sigma)|$ denoted by $l(\sigma)$ is called the conflict size. Let $\Pi^{\mathrm{i}}(S)$ denote the set of configurations in $\Pi(S)$ with $l(\sigma) = i$.

Let R be a random subset of S. We define a subspace $\Pi(R)$ as the set of all such configurations $\sigma \in \Pi(S)$ whose defining line segments are a subset of R. The conflict list in R for such configurations is $L(\sigma) \cap R$. Define $\Pi^*(R)$ to be the set of configurations in $\Pi(R)$ which are in the slabs formed by the pair of edges of the upper envelope of R scanned left to right. Clearly for a region $\sigma \in \Pi^*(R)$, the set of line segments of R, conflicting with σ is empty, i.e. $L(\sigma) \cap R$ is empty. That is $\Pi^*(R) \subseteq \Pi^o(R)$.

Clearly our configuration space $\Pi(S)$ has bounded valence, since the number of regions defined by the same set of line segments is bounded (constant). In fact this number is exactly equal to one. Thus the following random sampling results due to Clarkson and Shor hold for our problem.

Lemma 3 ([9,16]). *For some suitable constant k and large n,*

$$\Pr\left[\sum_{\sigma \in \Pi^*(R)} l(\sigma) \geq kn\right] \leq 1/c \ ,$$

for some constant $c > 1$, where probability is taken over all possible choices of the random sample R.

The above lemma gives a bound on the total size of the sub-problems.

Lemma 4 ([9,16]). *For some suitable constant k_1 and large n,*

$$\Pr\left[\max_{\sigma \in \Pi^*(R)} l(\sigma) \geq k_1 (n/r) \log r\right] \leq 1/c \ ,$$

for some constant $c > 1$, where probability is taken over all possible choices of random sample R such that $|R| = r$.

This lemma gives a bound on the maximum size of each sub-problem.

A sample is "good" if it satisfies the properties of Lemmas 3 and 4 simultaneously. Infact we have the following.

Lemma 5. *We can find a sample R which satisfies both Lemmas 3 and 4 simultaneously with high probability. Moreover, this can be done in $O(\log r)$ time and $O(n \log r)$ work with high probability.*

Proof. This can be done using Resampling and Polling [18]. □

We call a region "critical" if it contains at least one output vertex. Let $\Pi^*(R)$ denote the set of regions introduced by the random sample R and $\Pi_{\rm h}(R)$ denote the set of critical regions. Since $|\Pi_{\rm h}(R)| \leq H$, a good sample clearly satisfies the following property also.

Lemma 6. *For a good sample R, with $|R| = r$,*

$$\sum_{\sigma \in \Pi_{\rm h}({\rm R})} l(\sigma) = O(nH \log r/r) \ .$$

This will be used repeatedly in the analysis to estimate the non-redundant line segments whenever $H \leq r/\log r$.

We say that a line segment is redundant if it does not intersect any critical region, i.e. either it lies entirely below the upper envelope of R or it intersects some region lying above the upper envelope but not the critical region. Consider a region that does not contain any output vertex. Clearly only one line segment is useful in this region, which is the line segment that has the maximum y-coordinate in that region. Such a line segment must intersect at least one of the regions containing an output point and is therefore retained.

3.2 The Algorithm

Let $n_{\rm i}$ (respectively $r_{\rm i}$) denote the size of the problem (respectively sample size) at the $i^{\rm th}$ iteration with $n_1 = n$. Repeat the following procedure until $r_i > n^{\varepsilon}$ (this condition guarantees that the sample size is never too big) or $n_{\rm i} < n^{\varepsilon}$ for some fixed ε between 0 and 1. If $n_{\rm i} < n^{\varepsilon}$ then find out the upper envelope of $n_{\rm i}$ line segments directly using a brute force algorithm, else do one more iteration and find out $n_{\rm i+1}$. If $n_{\rm i+1} = O(n^{\varepsilon})$ then we use a brute force algorithm, otherwise we use the algorithm of Chen and Wada [7] to find the upper envelope of $n_{\rm i+1}$ line segments. The following is the description of the $i^{\rm th}$ iteration of the algorithm.

Rand-UE (i)

1. Choose a "good" sample R of size $r_{\rm i}$ = constant for $i = 1$ and $r^2_{\rm i-1}$ for $i > 1$. Find out the upper envelope of R.
2. (a) For every line segment find out the regions that it intersects.
   (b) Discard the redundant line segments lying below the upper envelope of R. (Sect. 4 discusses this in detail).
   (c) If the sum taken over all the remaining line segments, of the regions intersecting a line segment is $O(n)$ then continue else go to Step 1.
3. Filter out the segments not conflicting with any critical region as follows.
   (a) Compute $\Pi_{\rm h}(R)$ as follows.
   For every region σ do the following:
      i. Find out the line segments intersecting σ and assign as many processors to it (see Lemma 3 and Step 2(c) above).

ii. Consider the points of intersection of the line segments with the vertical lines defining the region (the boundaries of the slab lying above the edges of the upper envelope) including the intersection points by the edges of the upper envelope. If the points with maximum y-coordinate belong to the same line segment say s, and there does not exist any segment above s lying entirely in σ, then $\sigma \notin \Pi_{\mathrm{h}}(R)$ else $\sigma \in \Pi_{\mathrm{h}}(R)$.

(b) Delete a line segment if it does not belong to $\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)$.

4. The set of line segments for the next iteration is $\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)$ and its size is $n_{\mathrm{i}+1} = |\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)|$. Increment i and go to 1.

The analysis of Gupta and Sen [13] goes through here also. However detecting and deleting the redundant segments require further explanation which is done in Sect. 4. Hence we have the following results.

Lemma 7. *The upper envelope of n possibly intersecting straight line segments can be constructed in $O(\max\{\log H, \log\log n\})$ time with high probability using a linear number of processors.*

Remark 2. For $\log H = \Omega(\log\log n)$, this attains the ideal $O(\log H)$ running time with n processors, keeping the work optimal.

Using the standard slow-down method the algorithm is made optimal for all values of H. That is, we use $p = n/\rho$ processors where $\rho = \log\log n$, instead of n processors.

Theorem 2. *The upper envelope of n possibly intersecting straight line segments can be constructed in $O(\log H \cdot \log\log n)$ expected time and $O(n \log H)$ operations with high probability in a CRCW PRAM model where H is the size of the upper envelope.*

Proof. Refer to Gupta and Sen [13]. □

4 Finding the Redundant Line Segments

For the purpose of this section we call a segment redundant if it lies entirely below the upper envelope of the random sample. Otherwise, we call it non-redundant. Consider an upper envelope of the random sample of line segments, divided into regions (as defined in Sect. 3.1). If a line segment intersects the upper envelope at any point then it is non-redundant. However, if it does not intersect, then two cases arise: 1. Either it lies entirely above the upper envelope in which case, it is non-redundant or, 2. It lies entirely below the upper envelope in which case, it is redundant. So to determine if the line segment is redundant or not, we need to determine if it intersects the upper envelope. Let the vertices of the upper envelope spanned by a line segment be called the defining points for that line segment. If a line segment intersects the part of the envelope defined only by the

defining points, then the line segment intersects the upper envelope and hence, it is declared non-redundant. But if it does not intersect this part, then if it lies above the upper envelope of its defining vertices, then it is non redundant. Else, if it lies below the upper envelope of its defining vertices, we consider the two extreme slabs which the line segment intersects. If the line segment intersects the part of the envelope in these extreme slabs then it is non-redundant, else it is redundant. Thus determining whether a line segment is redundant or not reduces to determining whether it intersects the upper envelope formed by its defining vertices.

Lemma 8. *If a line segment intersects the lower convex chain of its defining vertices then it intersects the upper envelope defined by them.*

Proof. Ommitted. □

The task of determining whether a segment is redundant or not, reduces to finding whether it intersects the lower convex chain of its defining vertices. Checking whether it lies above or below, in case it doesn't, can be done by checking it against one of the vertices of the lower convex chain in constant time. Clearly, a segment that does not intersect the lower convex chain and lies above it is non-redundant (whether it intersects the upper envelope or not). For the segment that does not intersect the lower convex chain and lies below it, we will examine the extreme slabs. Checking the extreme slabs can be done in constant time because we need to consider only two regions having a constant number of edges; check whether the line segment intersects any of these edges. The case when a line segment belongs entirely to one region is a special one in which the two extreme regions coincide. Hence, this again can be handled in constant time.

4.1 The Locus Based Approach

We use a locus-based approach to find whether the line segment is redundant. We define an equivalence relation where two segments are equivalent if and only if they have the same set of defining points. The number of such equivalence classes is $O(m^2)$. For each of these equivalence classes we take a representative line segment and precompute the lower convex chain of its defining vertices. Using a brute force algorithm we can compute the lower convex chain of the vertices associated with all the equivalence classes in constant time using $O(m^6)$ processors. Hence we have the following lemma.

Lemma 9. *Given an upper envelope of size m, we can preprocess its vertices in constant time using $O(m^6)$ processors, such that given any arbitrary line segment with k processors, we can determine whether it is redundant or not in $O(\log m/\log k)$ time.*

Proof. Since convex chains have been precomputed, the result follows from the fact that, given a convex chain C of size c and a line segment 'l', whether the line segment intersects C can be determined in $O(\log c)$ time or in $O(\log c/\log k)$ time by a k-way search using k processors. □

5 The $\tilde{O}(\log n/\log k)$ Time Algorithm

The general technique given by Sen [21] to develop sub-logarithmic algorithms can be used to design an $O(\log n/\log k)$ algorithm for the problem. Let the number of line segments be n and the number of processors be (nk), $k > \log^{\Omega(1)} n$.

1. Pick up a "good" sample R of size $r = nk^{1/4}$.
2. Compute the upper envelope of the sample using a brute force algorithm.
3. Remove all the redundant line segments (those line segments that lie completely below the upper envelope of the sample computed in Step 2).
4. For every remaining line segment find out the number of regions that it intersects. If the sum taken over all the line segments is $O(n)$ then continue else goto Step 1.
5. For every region σ do the following
   (a) Find out the line segments intersecting the region and assign k times as many processors to it. (see Lemma 3, 4 and Step 4).
   (b) If the size of the sub problem is $> \log^{\Omega(1)} n$, then recurse, else
   (c) Solve the sub-problem in constant time. With $k > \log^{\Omega(1)} n$ this can be done in constant time.
6. Merge the upper envelopes formed in all the sub problems.

Theorem 3. *Given n straight line segments and nk $(k > \log^{\Omega(1)} n)$ processors, we can find their upper envelope in expected $O(\log n/\log k)$ time with high probability.*

Proof. Refer to Sen [21]. □

6 Remarks and Open Problems

One of the issues that remains to be dealt with is that of speeding up the algorithm further using a superlinear number of processors such that the time taken is $\Omega(\log H/\log k)$ with nk processors, where $k > 1$. Designing an algorithm that takes $O(\log H)$ time and does optimal work for all values of H is another open problem.

The other issue is to design a sub-logarithmic time algorithm using superlinear number of processors for $k > 1$. The technique of Sen [18,21] to filter the redundant line segments to control the blowup in the problem size and the processor allocation scheme used by them is not particularly effective here. In that, we allocate one processor for every output vertex contributed by the segment. As the number of output vertices contributed by a segment can't be predicted, we don't know how many processors to allocate to a given line segment and the requirement of a segment as it goes down the levels of recursion may increase.

References

1. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C. K. Yap, Parallel computational geometry. in: Proc. 25^{th} Annual Sympos. Found. Comput. Sci. (1985) 468-477; also: Algorithmica 3 (3) (1988) 293-327.
2. P. K. Aggarwal and M. Sharir, Davenport-Schinzel Sequences and their Geometric Applications. (1995).
3. S. Akl, Optimal algorithms for computing convex hulls and sorting, Computing 33 (1) (1984) 1-11.
4. H. Bast and T. Hagerup, Fast parallel space allocation, estimation and integer sorting, Technical Report MPI-I-93-123 (June 1993).
5. B. K. Bhattacharya and S. Sen, On simple, practical, optimal, output-sensitive randomized planar convex hull algorithm, J. Algorithms 25 (1997) 177-193.
6. B. Chazelle and J. Fredman, A deterministic view of random sampling and its use in geometry, Combinatorica 10 (3) (1990) 229-249.
7. W. Chen and K. Wada, On computing the upper envelope of segments in parallel, Proc. International Conference on Parallel Processing. (1998).
8. K. L. Clarkson, A randomized algorithm for computing a face in an arrangement of line segments.
9. K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387-421.
10. M. Goodrich, Geometric partitioning made easier, even in parallel, in: Proc. 9^{th} ACM Sympos. Comput. Geom. (1993) 73-82.
11. L. J. Guibas, M. Sharir, and S. Sifrony, On the general motion planning problem with two degrees of freedom, Discrete Comput. Geom., 4 (1989), 491-521
12. N. Gupta, Efficient Parallel Output-Size Sensitive Algorithms, Ph.D. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Delhi (1998).
13. N. Gupta and S. Sen, Optimal, output-sensitive algorithms for constructing planar hulls In parallel, Comput. Geom. Theory and App. 8 (1997) 151-166.
14. N. Gupta and S. Sen, Faster output-sensitive parallel convex hull for $d \leq 3$: optimal sub-logarithmic algorithms for small outputs, in: Proc. ACM Sympos. Comput. Geom. (1996) 176-185.
15. J. Hershberger, Finding the upper envelope of n line segments in $O(n \log n)$ time, Inform. Process. Lett. 33 (1989) 169-174.
16. M. Ketan, Computational Geometry: An Introduction through Randomized Algorithms. Prentice Hall, Englewood Cliffs, NJ, (1994).
17. F.Neilsen and M.Yvinec, An output-sensitive convex hull algorithm for convex objects. Institut National De Recherche En Informatique Et En Automatique. (1995).
18. S. Rajasekaran and S. Sen, in: J. H. Reif, Ed., Random Sampling Techniques and Parallel Algorithm Design (Morgan Kaufmann Publishers, San Mateo, CA, 1993).
19. J. H. Reif and S. Sen, Randomized algorithms for binary search and Load Balancing on fixed connection networks with geometric applications, SIAM J. Comput. 23 (3) (1994) 633-651.
20. J. H. Reif and S. Sen, optimal parallel randomized algorithms for 3-D convex hull, and related problems, SIAM J. Comput. 21 (1992) 466-485.
21. S. Sen, Lower bounds for parallel algebraic decision trees, parallel complexity of convex hulls and related problems, Theoretical Computer Science. 188 (1997) 59-78.

List Decoding from Erasures: Bounds and Code Constructions

Venkatesan Guruswami*

University of California at Berkeley
Computer Science Division
Berkeley, CA 94720.
venkat@theory.lcs.mit.edu

Abstract. We consider the problem of list decoding from erasures. We establish lower and upper bounds on the rate of a (linear) code that can be list decoded with list size L when up to a fraction p of its symbols are adversarially erased. Our results show that in the limit of large L, the rate of such a code approaches the capacity $(1-p)$ of the erasure channel. Such nicely list decodable codes are then used as inner codes in a suitable concatenation scheme to give a uniformly constructive family of asymptotically good binary linear codes of rate $\Omega(\varepsilon^2/\lg(1/\varepsilon))$ that can be efficiently list decoded using lists of size $O(1/\varepsilon)$ from up to a fraction $(1-\varepsilon)$ of erasures. This improves previous results from [14] in this vein, which achieved a rate of $\Omega(\varepsilon^3 \lg(1/\varepsilon))$.

Keywords: Error-correcting codes, Linear codes, Decoding algorithms, List decoding, Erasure channel.

1 Introduction

The list decoding problem for an error-correcting code consists of outputting a list of all codewords that lie within a certain Hamming distance of the received word [6,19]. The decoding is considered successful as long as the correct codeword is included in the list. If the decoder is restricted to output lists of size L, the decoding is called list-of-L decoding. List decoding with even moderate-sized lists allows one to correct significantly more errors than possible with unambiguous decoding (cf. [7,13,11]).

In this paper, we consider the problem of list decoding from *erasures*, i.e. list decoding when a certain fraction of the symbols of the received word are *adversarially* erased. Specifically, we are interested in codes with non-trivial list decoding performance – i.e. codes of "large" rate that are list decodable using "small" lists from a "large" number of erasures. Such a study has already been carried out with some success for the case of errors [7,20,4,11]. Though the case of erasures is algorithmically always easier to handle, it raises several interesting

* Work done while the author was at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 195–206, 2001.
© Springer-Verlag Berlin Heidelberg 2001

combinatorial questions which this paper addresses. Also, the simpler situation of erasures makes better tradeoffs (between rate and list decodability) possible than for the errors case. For example, for binary codes the maximum fraction of errors one can hope to correct from is $(1/2 - \varepsilon)$, where as decoding up to a fraction $(1 - \varepsilon)$ of erasures is possible. The rate (as a function of ε) that one can achieve for the erasures case also turns out be better than what is possible for the errors case.

1.1 Definitions and Notation

We focus on linear codes throughout this paper. For simplicity we also restrict our attention to binary codes though all our results go through for larger alphabets as well. Recall that a binary linear code C of blocklength n, dimension k and minimum distance d, denoted an $[n, k, d]_2$ code, is a k-dimensional subspace of $\mathbb{F}_2^n$ such that the minimum Hamming weight of a non-zero element (called codeword) of C equals d. When we omit the distance, we refer to the code as an $[n, k]_2$ code. The rate of the code is defined to be the ratio k/n and the relative distance is defined to be d/n. The main thrust of this paper is the asymptotic performance of codes, and we therefore focus on infinite families $\mathcal{C} = \{C_i\}_{I \geq 1}$ of $[n_i, k_i, d_i]_2$ codes of increasing blocklength $n_i \to \infty$. The rate $R(\mathcal{C})$ of such a family is defined to be $R(\mathcal{C}) \stackrel{\text{def}}{=} \liminf_i \left\{ \frac{k_i}{n_i} \right\}$ and its relative distance is defined to be $\delta(\mathcal{C}) \stackrel{\text{def}}{=} \liminf_i \left\{ \frac{d_i}{n_i} \right\}$.

We next define the list decoding radius for recovering from erasures. For $y \in \{0,1\}^n$ and $T \subseteq \{1, 2, \ldots, n\}$, define $[y]_T \in \{0,1\}^{|T|}$ to be the projection of y onto the coordinates in T. A code C of blocklength n is said to be (s, L)-*erasure list-decodable* if for every $r \in \{0,1\}^{n-s}$ and every set $T \subseteq \{1, 2, \ldots, n\}$ of size $(n-s)$, we have $|\{c \in C : [c]_T = r\}| \leq L$. In other words, given any received word with at most s erasures, the number of codewords consistent with the received word is at most L. Note that a code of minimum distance d is $(d-1, 1)$-erasure list-decodable, but is *not* $(d, 1)$-erasure list-decodable.

For linear codes, list decoding from erasures is algorithmically easy as it just amounts to finding the space of solutions to a linear system. Thus, if a linear code is (s, L)-erasure list-decodable for some small L, then a list of size at most L can also be *efficiently output* given any received word with at most s erasures. (This is not true for the case when there are errors, where algorithmic list decodability is potentially a lot more difficult to achieve than combinatorial list decodability – in fact there are known constructions of codes with good combinatorial list decodability, but for which efficient list decoding algorithms are not known.) Since our focus in this paper is on recovery from erasures, we only focus on linear codes with good combinatorial erasure list decodability, and this automatically implies a polynomial time list decoding algorithm from a large number of erasures. But for our main explicit code construction, we do mention how the decoding time can be improved beyond the time required to solve a linear system.

Definition 1 (Erasure List Decoding Radius) *For an integer $L \geq 1$ and a code C, the* list-of-L erasure decoding radius *of C, denoted* $\text{radius}_L(C)$*, is defined to be the maximum value of s for which C is (s, L)-erasure list-decodable.*

Definition 2 (ELDR for Code Families) *For an infinite family $\mathcal{C} = \{C_i\}_{i \geq 1}$ of codes and an integer $L \geq 1$, the* list-of-L erasure decoding radius *of $\mathcal{C}$, denoted* $\mathsf{ELDR}_L(\mathcal{C})$*, is defined to be*

$$\mathsf{ELDR}_L(\mathcal{C}) \stackrel{\text{def}}{=} \liminf_i \left\{ \frac{\text{radius}_L(C_i)}{n_i} \right\} , \tag{1}$$

where n_i is the blocklength of C_i.

We now define the function which measures the trade-off achievable between rate and erasure list decoding radius.

Definition 3 *For an integer L and $0 \leq p \leq 1$, the maximum rate of a code family with list-of-L erasure decoding radius at least p, denoted $R_L(p)$, is defined as*

$$R_L(p) \stackrel{\text{def}}{=} \sup_{\mathcal{C}:\ \mathsf{ELDR}_L(\mathcal{C}) \geq p} R(\mathcal{C}) . \tag{2}$$

1.2 Main Results

Combinatorial Bounds. One of the objectives of this paper is to establish lower and upper bounds on this function $R_L(p)$ (Theorem 1 and Theorem 2). Since the list-of-1 ELDR of a code family equals its relative distance δ, $R_1(p) = R(\delta)$ is the central function in coding theory that studies the trade-off between the rate and minimum distance of a code. One of the consequences of results of this paper is that in the limit of large L, the function $R_L(p)$ tends to $1-p$, thus matching the singleton bound. This result has the following nice interpretation. It is a classical result in coding theory that the capacity of the erasure channel where each codeword symbol is randomly and independently erased with probability p, equals $(1-p)$ [5]. Thus our results show that using list decoding with large enough (but constant-sized) lists, one can approach capacity even if the symbols that are erased are *adversarially* (and not randomly) chosen. Our upper bound on $R_L(p)$ also shows that $R_L(p) < 1-p$ for every p and every fixed list size L (this result holds even for general, non-linear codes). Thus one *needs* unbounded list size in order to approach the capacity of the adversarial erasure channel using list decoding. We remark that analogous statements also hold for the errors case from earlier results in [7,20,4,11].

Code Constructions. For any constant $\varepsilon > 0$, our lower bound on $R_L(p)$ implies the existence of a binary linear code family $\mathcal{C}_\varepsilon = \{C_i\}_{i \geq 1}$ of rate $\Omega(\varepsilon / \lg(1/\varepsilon))$ such that each C_i is $((1-\sigma)n_i, O(1/\sigma))$-erasure list-decodable for every σ satisfying $\varepsilon \leq \sigma \leq 1$. One can use such codes as inner codes in a concatenated scheme

with outer code being any code of rate $\Omega(\varepsilon)$ and relative distance $1 - O(\varepsilon)$ (like Reed-Solomon codes, or, if one wants codes over constant alphabet size, algebraic-geometric codes or the expander based codes of [2]). Such a concatenation scheme gives a *uniformly constructive* family of binary linear codes of rate $\Omega(\varepsilon^2/\lg(1/\varepsilon))$ that can be list decoded from a fraction $(1-\varepsilon)$ of erasures using lists of size $O(1/\varepsilon)$.[1] The previous best result in this vein from [14] achieved a rate of $\Omega(\varepsilon^3)$, by exploiting the fact that any family of codes with relative distance greater than $(1-\varepsilon)/2$ has a small list size for decoding from $(1-\varepsilon)$ erasures. [2] The best construction of such high distance codes achieves a rate of $\Omega(\varepsilon^3)$ (see [2]). We stress that the novelty of our result is that it is **not** obtained by appealing to this distance to ELDR relation, and indeed no polynomial time constructions of binary code families of relative distance $(1/2 - O(\varepsilon))$ and rate about ε^2 are known. In fact such a construction (which will asymptotically match the Gilbert-Varshamov bound at low rates) will be a *major* breakthrough in coding theory.

Moreover, by using Reed-Solomon codes as outer codes, we can get a code with slightly worse construction time, but which can be encoded in near-linear time and decoded in near-quadratic time. The decoding is based on a list decoding algorithm due to [13] and its fast implementation due to [16,17].

2 Combinatorial Bounds on $R_L(p)$

2.1 Lower Bound on $R_L(p)$

All logarithms in this paper are to the base 2 and will be denoted by $\lg x$. The following folklore combinatorial lemma gives a useful linear-algebraic characterization of when a linear code is (s, L)-erasure list-decodable.

Lemma 1. *An $[n,k]_2$ linear code C is (s,L)-erasure list-decodable if and only if its $n \times k$ generator matrix G has the property that every $(n-s) \times k$ submatrix of G has rank at least $(k - \lfloor \lg L \rfloor)$.*

Proof: Let $T \subseteq \{1, 2, \ldots, n\}$ with $|T| = n - s$ and $r \in \{0,1\}^{n-s}$, the number of codewords $c \in C$ with $[c]_T = r$ is precisely the number of solutions $x \in \{0,1\}^k$ to the system $G_T x = r$ where G_T is the submatrix of G consisting of all rows indexed by elements in T. By standard linear algebra, the number of solutions x to the linear system $G_T x = \mathbf{0}$ is precisely 2^ℓ where $\ell = k - \mathrm{rank}(G_T)$, and for any $r \in \{0,1\}^{n-s}$, the number of solutions x to $G_T x = r$ is at most 2^ℓ (in fact, it is always either 0 or 2^ℓ). Hence C is (s,L)-erasure list-decodable if and only if for every $T \subseteq \{1, \ldots, n\}$ with $|T| = n - s$, G_T has rank at least $k - \lfloor \lg L \rfloor$. □

[1] The term "uniformly constructive" means a polynomial time construction where the exponent of the polynomial is an absolute constant independent of ε.

[2] More generally, any binary code of relative distance δ has a small (linear in block-length) number of codewords for up to a fraction 2δ of erasures, and this is the best possible in the sense that for every $\gamma > 0$, there exist linear codes C_γ of relative distance δ, and a received word y with a fraction $(2+\gamma)\delta$ of erasures, such that C_γ has exponentially many codewords consistent with y.

Theorem 1. *For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, we have*

$$R_L^{\mathrm{lin}}(p) \geq 1 - p - \frac{H(p)}{\lceil \lg(L+1) \rceil} \ . \tag{3}$$

The above theorem follows from Part (i) of the stronger claim of the lemma below. The second part of the lemma will be used in the application to concatenated codes in Section 3. It asserts the existence of codes that exhibit a "gradual" increase in list size as more and more codeword symbols are erased, up to a total of $(1-\varepsilon)n$ erasures.

Lemma 2. *For all large enough n, the following hold:*

(i) For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, there exists an $[n,k]_2$ linear code C with $k = \lfloor \left(1 - p - \frac{H(p)}{\lceil \lg(L+1) \rceil}\right) n - \sqrt{n} \rfloor$ that is (pn, L)-erasure list-decodable.

(ii) There exists absolute constants $A, B > 0$ such that for every ε, $0 \leq \varepsilon \leq 1$, there exists an $[n,k]_2$ linear code C with $k = \lfloor \frac{\varepsilon n}{\lg(A/\varepsilon)} \rfloor$ that is $(s, \frac{Bn}{n-s})$-erasure list-decodable for every *$s \leq (1-\varepsilon)n$.*

Proof: We first prove Part (i) of the lemma. The proof is based on the probabilistic method. We will pick a code C generated by a random $n \times k$ generator matrix G where k is as specified in the statement of the lemma.[3] We will prove that such a random code is (pn, L)-erasure list-decodable with high probability. By Lemma 1, this amounts to estimating the probability that a random $n \times k$ matrix over $\mathbb{F}_2$ has the property that every $t \times k$ submatrix has rank at least $(k - \lfloor \lg L \rfloor)$, where $t = (1-p)n$. For any J', $0 \leq J' \leq k$, the probability that a random matrix M of order $t \times k$ has rank $(k - J')$ is at most $2^{kJ'} 2^{-tJ'}$. This follows since for a fixed subspace S of $\mathbb{F}_2^k$ of dimension $(k - J')$, the probability that all t rows of M lie in S is at most $2^{-J't}$, and furthermore the number of subspaces of $\mathbb{F}_2^k$ of dimension $(k - J')$ is at most $2^{kJ'}$ as one can specify such a subspace as the null-space of a $J' \times k$ matrix over $\mathbb{F}_2$. Let $J \stackrel{\mathrm{def}}{=} \lceil \lg(L+1) \rceil = 1 + \lfloor \lg L \rfloor$. Then, for $t \geq k$, the probability that a random matrix M of order $t \times k$ has rank $(k - J)$ or less is at most

$$\sum_{J'=J}^{k} 2^{(k-t)J'} \leq k 2^{(k-t)J} \ .$$

Now, by a union bound, the probability that some $t \times k$ submatrix of G has rank at most $(k - J)$ is at most

$$\begin{aligned} \binom{n}{t} \cdot k 2^{(k-t)J} &= \binom{n}{(1-p)n} \cdot k 2^{(k-t)J} \\ &\leq k \cdot 2^{H(p)n} \cdot 2^{kJ - (1-p)nJ} \end{aligned}$$

[3] We will assume for simplicity that C has dimension k, i.e. G has full column rank, since this happens with very high probability.

$$\begin{aligned}
&\leq k \cdot 2^{H(p)n} \cdot 2^{\left((1-p)J - H(p)\right)n} \cdot 2^{-J\sqrt{n}} \cdot 2^{-(1-p)nJ} \\
&= k \cdot 2^{-J\sqrt{n}} \\
&= o(1) \ ,
\end{aligned}$$

where in the second step we have used the fact that $\binom{n}{(1-p)n} = \binom{n}{pn} \leq 2^{H(p)n}$ (cf. [18, Chap. 1, Theorem 1.4.5]), and in the third step we substituted the value of k from the statement of Part (i) of the lemma.

Hence an $n \times k$ matrix G every $t \times k$ submatrix of which has rank at least $k - J + 1 = k - \lfloor \lg L \rfloor$ exists, and by Lemma 1, therefore, there exists a (pn, L)-erasure list-decodable code of blocklength n and dimension k. This proves Part (i) of the lemma.

To prove Part (ii), we apply the above proof for $p = (1 - \varepsilon)$, and $k = \lfloor \frac{\varepsilon}{\lg(8/\varepsilon)} n \rfloor$. Let us estimate the probability that for a fixed s, where $n/2 \leq s \leq (1 - \varepsilon)n$, C is not $(s, \frac{8n}{n-s})$-erasure list-decodable. By Lemma 1, this happens only if some $(n - s) \times k$ submatrix of G has rank less than $k - \lg(\frac{8n}{n-s})$. Let $\sigma = (n - s)/n$; we have $\varepsilon \leq \sigma \leq 1/2$. As in the proof of Part (i), the probability that some $(n - s) \times n$ submatrix of G has rank less than $k - \lg(8/\sigma)$ is at most

$$\begin{aligned}
\binom{n}{n-s} \cdot k \cdot 2^{(k-n+s)\lg(8/\sigma)} &\leq k \cdot \left(\frac{e}{\sigma}\right)^{\sigma n} \cdot 2^{\left((\varepsilon/\lg(8/\varepsilon)) - \sigma\right)\lg(8/\sigma)n} \\
&= k \cdot 2^{-n\left(\sigma \lg(8/e) - \frac{\varepsilon \lg(8/\sigma)}{\lg(8/\varepsilon)}\right)} \\
&= 2^{-\Theta(n)}
\end{aligned}$$

where the last step follows since $\sigma \geq \varepsilon$, and in the first step we used the inequality $\binom{n}{\sigma n} \leq (e/\sigma)^{\sigma n}$ for $\sigma \leq 1/2$.

Now, by a union bound, the probability that for some s, $n/2 \leq s \leq (1-\varepsilon)n$, C is not $(s, \frac{8n}{n-s})$-erasure list-decodable is also exponentially small. Hence there exists a linear code C of blocklength n and rate $\varepsilon/\lg(8/\varepsilon)$ that is $(s, \frac{8n}{n-s})$-erasure list-decodable for every s that satisfies $n/2 \leq s \leq (1 - \varepsilon)n$. Since the list size for $s < n/2$ erasures is at most the list size for $n/2$ erasures, such a code is also $(s, \frac{16n}{n-s})$-erasure list-decodable for every s, $0 \leq s \leq (1 - \varepsilon)n$. This proves Part (ii) of the claim (with the choice $A = 8$ and $B = 16$ for the absolute constants). □

Remark: Note that the above lemma not only guarantees the existence of codes with the required properties, but also proves that a random code has these properties with very high probability.

2.2 Upper Bound on $R_L(p)$

We now turn to upper bounds on $R_L(p)$. It is easy to prove that for any fixed L (in fact even for a list size L that is allowed to grow polynomially in the blocklength), we must have $R_L(p) \leq 1 - p$. Indeed, let C be a code of blocklength n and rate r, and let $T = \{1, 2, \ldots, (1 - p)n\}$. Pick a *random* $y \in \{0, 1\}^{(1-p)n}$

and consider the set S_y of codewords $c \in C$ that satisfy $[c]_T = y$. The expected number of such codewords equals $2^{rn}2^{-(1-p)n}$, and hence there must exist a $y \in \{0,1\}^n$ for which $|S_y| \geq 2^{(r-(1-p))n}$. Since we want $|S_y| \leq L \leq \text{poly}(n)$, we must have $r \leq (1-p)$. Hence $R_L(p) \leq 1-p$. Below, we are interested in a better upper bound on $R_L(p)$, which in particular bounds it strictly away from $(1-p)$ for every fixed L (and thereby shows that one requires unbounded list size in order to approach the "capacity" $(1-p)$ of the erasure channel). The proof uses standard techniques similar to those used in the Plotkin and Elias-Bassalygo bounds for the rate vs. minimum distance trade-off. It will appear in the full version of the paper and can also be found in [10, Chapter 10].

Theorem 2. *For every integer $L \geq 1$ and every p, $0 \leq p \leq 1 - 2^{-L}$, we have*

$$R_L(p) \leq \min\left\{1 - H(\lambda), 1 - \frac{p}{1-2^{-L}}\right\} , \tag{4}$$

where λ is the unique root of the equation $\lambda^{L+1} + (1-\lambda)^{L+1} = 1-p$ in the range $0 \leq \lambda \leq 1/2$. For $p \geq 1 - 2^{-L}$, we have $R_L(p) = 0$.

Corollary 3 *For every integer L and every p, $0 < p < 1$, $R_L(p) < 1 - p$.*

Remark: For $L = 1$, the two bounds proven in Theorem 2 are precisely the Elias-Bassalygo and Plotkin bounds on the rate of a code in terms of its minimum distance, which state that $R(\delta) \leq 1 - H(\frac{1-\sqrt{1-2\delta}}{2})$ and $R(\delta) \leq 1 - 2\delta$, respectively. For large L, the bound $R_L(p) \leq 1 - p/(1-2^{-L})$ is better than the other bound $R_L(p) \leq 1 - H(\rho)$ *except for* very small p (less than about $L/2^L$).

3 Application to Concatenated Codes

We now use the codes guaranteed by Lemma 2 (specifically Part (ii) of the lemma) as inner codes in a suitable concatenation scheme to construct binary code families of rate $\Omega(\varepsilon^2/\lg(1/\varepsilon))$ that can be efficiently list decoded using lists of size $O(1/\varepsilon)$ when up to a fraction $(1-\varepsilon)$ of its symbols are erased. This follows from the following theorem (Theorem 4), which is the main result of this section. The proof uses the simple, yet powerful, tool of code concatenation [8]. Given an $[N, K, D]_{2^m}$ code C_1, and an $[n, m, d]_2$ code C_2, the concatenated code $C_1 \oplus C_2$ is an $[Nn, Km, \geq Dd]_2$ binary linear code whose codewords are obtained by encoding a message x first using C_1, and then encoding each of the symbols of $C_1(x)$ using C_2 (note that each symbol of $C_1(x)$ is an element of $\text{GF}(2^m)$ and can hence be interpreted as an m-bit vector, and m is exactly the dimension of C_2).

Theorem 4. *For every $\varepsilon > 0$, there exists a family of binary linear codes of rate $\Omega(\varepsilon^2/\log(1/\varepsilon))$ such that a code of blocklength in the family can be constructed in $2^{\varepsilon^{-O(1)}} + \text{poly}(N, 1/\varepsilon)$ time, and can be list decoded in polynomial time using lists of size $O(1/\varepsilon)$ when up to a fraction $(1-\varepsilon)$ of its symbols are erased.*

The above result follows immediately from the following lemma.

Lemma 3. *There exist absolute constants b, d such that for all large enough integers K and all small enough $\varepsilon > 0$, there exists a binary linear code C_K that has the following properties:*

(i) C_K has dimension K and blocklength $N \leq \frac{bK \log(1/\varepsilon)}{\varepsilon^2}$.

(ii) The generator matrix of C_K can be constructed in $2^{\varepsilon^{-O(1)}} + \mathrm{poly}(N, 1/\varepsilon)$ time.

(iii) C_K is $((1-\varepsilon)N, \frac{d}{\varepsilon})$-erasure list-decodable (and since C_K is linear there is an $O(N^3)$ time list decoding algorithm to decode up to $(1-\varepsilon)N$ erasures using lists of size $O(1/\varepsilon)$).

Proof: The code C_K is constructed by concatenating an outer code C_{out} over $\mathrm{GF}(2^m)$ of blocklength n_0, dimension $k_0 = K/m$ and minimum distance d_0, with an inner code C_{in} as guaranteed by Part (ii) of Lemma 2. By using a construction in [2], we can pick parameters so that $k_0 = K/m = \Omega(\varepsilon n_0)$, $m = O(1/\varepsilon)$ and $d_0 = (1-O(\varepsilon))n_0$ (for convenience, we hide constants using big-Oh notation, but we stress that these are absolute constants that do not depend on ε).[4] We note that this choice of C_{out} is not linear over $\mathrm{GF}(2^m)$ (though it *is* additive), but a (succinct) description of C_{out} can be constructed in time $\mathrm{poly}(n, 1/\varepsilon)$. Moreover, after concatenation with an inner binary linear code, the overall concatenated code will be a binary linear code.

The inner code C_{in} will be a code as guaranteed by Lemma 2 of dimension m and blocklength $n_1 = O(\frac{m \lg(1/\varepsilon)}{\varepsilon})$ that is $((1-\sigma)n_1, B/\sigma)$-erasure list-decodable for every $\sigma \geq \varepsilon/2$. We can construct such a code by a brute-force search in $2^{O(mn_1)} = 2^{\varepsilon^{-O(1)}}$ time. Once we have descriptions of C_{out} and C_{in}, one can construct the generator matrix of the concatenated code C_K in $\mathrm{poly}(N, 1/\varepsilon)$ time where $N = n_0 n_1$ is the blocklength of C_K. Now $N = n_0 \cdot n_1 = O(\frac{K}{m\varepsilon}) \cdot O(\frac{m \lg(1/\varepsilon)}{\varepsilon}) = O(\frac{K \lg(1/\varepsilon)}{\varepsilon^2})$. This proves Parts (i) and (ii) of the statement of the lemma.

We now prove that C_K is $((1-\varepsilon)N, O(1/\varepsilon))$-erasure list-decodable. Let y be a received word with $(1-\varepsilon)N$ erasures, and let $c'_1, c'_2, \ldots, c'_M$ be the codewords in C_K "consistent" with y, and let c_j be the codeword of C_{out} corresponding to c'_j, for $1 \leq j \leq M$. We wish to prove that $M = O(1/\varepsilon)$. Let y_i be the portion of y corresponding to the i'th outer codeword position, for $1 \leq i \leq n_0$. Let the number of symbols in y_i be $\sigma_i n_1$. We have $\sum_i \sigma_i = \varepsilon n_0$. Define $Q = \{i : \sigma_i \geq \varepsilon/2\}$. Clearly we have

$$\sum_{i \in Q} \sigma_i \geq \frac{\varepsilon n_0}{2} \ . \tag{5}$$

[4] Actually, we can use certain families of algebraic-geometric codes and even have $m = O(\log(1/\varepsilon))$. But we prefer to use the codes from [2] as they are easier to construct and have lower construction complexity, and we do not want to give the impression that we need complicated AG-codes for our construction. We will highlight the improvement in performance attainable using AG-codes in Section 3.2.

Now define "weights" $w_{i,\beta}$ for $1 \leq i \leq n_0$ and $\beta \in \mathrm{GF}(2^m)$ as follows. If $i \notin Q$, set $w_{i,\beta} = 0$ for all $\beta \in \mathrm{GF}(2^m)$. If $i \in Q$, then $\sigma_i \geq \varepsilon/2$ and hence by construction C_{in} is $((1-\sigma_i)n_1, B/\sigma_i)$-erasure list-decodable. In other words, if $J_i = \{\beta : C_{\mathrm{in}}(\beta)$ is consistent with $y_i\}$, then $|J_i| \leq B/\sigma_i$. We set (for $i \in Q$):

$$w_{i,\beta} = \begin{cases} \sigma_i & \text{if } \beta \in J_i \\ 0 & \text{if } \beta \notin J_i \end{cases}$$

Since $|J_i| \leq B/\sigma_i$, our choice of weights clearly satisfy

$$\sum_{i,\beta} w_{i,\beta}^2 \leq B \sum_{i \in Q} \sigma_i \ . \tag{6}$$

We now use a combinatorial result that gives an upper bound on the number of codewords of C_{out} that satisfy a certain weighted condition depending on the distance d_0 of C_{out}. This result is a "weighted" version of the classical "Johnson bounds", and appears, for example, in [15] and [10, Section 3.4]. Specifically, this result states that, for any $\varepsilon' > 0$, the number of codewords $(\alpha_1, \alpha_2, \ldots, \alpha_{n_0}) \in C_{\mathrm{out}}$ that satisfy

$$\sum_{i=1}^{n_0} w_{i,\alpha_i} \geq \Big(\big(n_0 - d_0(1-\varepsilon')\big) \sum_{i,\beta} w_{i,\beta}^2 \Big)^{1/2} \tag{7}$$

is at most $1/\varepsilon'$.

Now for each c_j, $1 \leq j \leq M$, and each $i \in Q$, we have $c_{j,i} \in J_i$, where $c_{j,i} \in \mathrm{GF}(2^m)$ denotes the i'th symbol of c_j. Now, $w_{i,c_{j,i}} = \sigma_i$ for every $i \in Q$ and $w_{i,c_{j,i}} = 0$ for $i \notin Q$. Thus, we have

$$\sum_{i=1}^{n_0} w_{i,c_{j,i}} = \sum_{i \in Q} \sigma_i \ . \tag{8}$$

Combining Equations (5), (6) and (8), we have that the codeword c_j satisfies Condition (7) as long as

$$\frac{\varepsilon n_0}{2} \geq (n_0 - d_0(1-\varepsilon'))B \ ,$$

which can be satisfied provided $d_0 \geq n_0\big(1 - \frac{\varepsilon}{2B}\big)(1-\varepsilon')^{-1}$. Picking $\varepsilon' = \frac{\varepsilon}{4B}$, we only need $d_0 = n_0(1 - O(\varepsilon))$ which is satisfied for our choice of C_{out}. Hence the number of codewords consistent with the received word y is at most $1/\varepsilon' = O(1/\varepsilon)$, thus proving Part (iii) of the lemma as well. □ *(Lemma 3)*

Remark: The previous results from [14] (obtained using code constructions from [2,3]) along the lines of the above theorem achieved a blocklength of $N = \min\{O(\frac{K^2}{\varepsilon^2}), O(\frac{K}{\varepsilon^3})\}$. Thus, our result achieves the best features of both these bounds and gets $N = \tilde{O}(K/\varepsilon^2)$ (hiding the $\lg(1/\varepsilon)$ factor).

3.1 Obtaining Near-Linear Encoding and Quadratic Decoding Times

The codes constructed in Lemma 3 being linear can be encoded in $O(N^2)$ time and can be list decoded in $O(N^3)$ time by solving a linear system. By using Reed-Solomon codes as outer code, the encoding can be accomplished in $\tilde{O}(N)$ time (here $\tilde{O}(N)$ denotes $O(N\mathrm{poly}(\lg N))$). This is because the outer encoding can be performed in $\tilde{O}(N)$ using FFT based methods and then the inner encoding just takes $O(\lg^2 N)$ time for each of the N outer codeword symbols. For decoding, the inner codes can be decoded in $O(\lg^3 N)$ time by solving a linear system, and then the Reed-Solomon code can be decoded from a set of weights that satisfy Condition (7) using a near-quadratic time implementation (cf. [16,17]) of the weighted Reed-Solomon decoding algorithm from [13]. However, the code family is no longer uniformly constructive as one must now search and find a good inner code of dimension $\Omega(\lg N)$ as opposed to the construction in Lemma 3 where the inner code has dimension which is a constant dimension depending only on ε. In fact, a naive search for the inner code among all inner codes will take time which is quasi-polynomial in N. However, one can use the method of conditional expectations to achieve a deterministic construction in $N^{O(\log^2(1/\varepsilon))}$ time — we omit the details.

3.2 Improving Construction Time Using Algebraic-Geometric Codes

Despite the impressive encoding and decoding times, the drawback of the above construction with Reed-Solomon codes as the outer code is the rather high exponent of N in the construction time. In particular, the code family ceases to be *uniformly constructive*, since the construction time is no longer of the form $O(f(\varepsilon)N^a)$ where a is an absolute constant independent of ε.

Instead of Reed-Solomon codes, we can use AG-codes of distance $(1 - O(\varepsilon))$ as the outer code. Such codes of rate $\Omega(\varepsilon)$ exist over an alphabet size of $O(1/\varepsilon^2)$ (any family of AG-codes that meet the so-called Drinfeld-Vlăduţ bound will satisfy this, cf. [10, Section 6.3.9]). Now, the dimension k of the inner binary code is only $O(\log(1/\varepsilon))$, and one can deterministically find a binary linear inner code which has the required erasure list decodability properties in $2^{O(\log^3(1/\varepsilon))}$ time.

Since the overall code is linear, the claims of quadratic encoding time and cubic decoding time still holds. Thus, we can obtain the same properties as the codes from Lemma 3 with an improved construction time of $2^{O(\log^3(1/\varepsilon))} + \mathrm{poly}(N, 1/\varepsilon)$. We formally record this improvement below.

Theorem 5. *For every $\varepsilon > 0$, there exists a family of binary linear codes of rate $\Omega(\varepsilon^2/\log(1/\varepsilon))$ such that a code of blocklength in the family can be constructed in $2^{\varepsilon^{-O(1)}} + \mathrm{poly}(N, 1/\varepsilon)$ time, and can be list decoded in polynomial time using lists of size $O(1/\varepsilon)$ when up to a fraction $(1 - \varepsilon)$ of its symbols are erased.*

4 Concluding Remarks

Our lower bound on $R_L(p)$ from Theorem 1 guarantees the *existence* of binary linear code families of rate $\Omega(\varepsilon)$ which can be list decoded from up to a fraction $(1-\varepsilon)$ of erasures, using lists of size, say, ε^{-2}. The construction of Theorem 4, however, only achieves a rate of about ε^2. It was recently observed in [12] that the result of Theorem 1 is actually "tight" in the "high-noise regime"– specifically they show that for linear codes, in order to have positive rate, one requires a list size of at least $\Omega(1/\varepsilon)$ to list decode from $(1-\varepsilon)$ erasures. This indicates that our results cannot be improved by the choice of a better linear code as inner code.

Furthermore, as pointed out to us by Alon [1] (see also the formal discussion in [10, Section 10.6.4]), getting a poly$(N, 1/\varepsilon)$ time construction of a binary *linear* code family of rate better than ε^2 (like $\varepsilon^{3/2}$, for example) with poly$(1/\varepsilon)$-sized lists for up to a fraction $(1-\varepsilon)$ of erasures, would imply a significant improvement for explicit constructions of certain bipartite Ramsey graphs. In this sense, improving our results further seems to be a significant challenge: there is, however, some hope since we allow for a construction that runs in time exponential in $1/\varepsilon$ and/or a list size that is exponential in $1/\varepsilon$. It is an interesting open question whether this can be exploited to beat the "ε^2 barrier" for the rate.

If one allows larger alphabets, known constructions of large distance algebraic-geometric codes imply code families of ELDR $(1-\varepsilon)$ and rate $\Omega(\varepsilon)$ over an alphabet of size $O(1/\varepsilon^2)$, even for a list size of 1. Explicit constructions of linear codes of rate better than $\Omega(\varepsilon^2)$ with ELDR $(1-\varepsilon)$ even for a slightly smaller alphabet do not appear to be known. However, in recent work, [12] present constructions of *non-linear* codes over large alphabets (of size *independent* of ε) that can be efficiently list decoded from a fraction $(1-\varepsilon)$ of erasures and which have rate better than $\Omega(\varepsilon^2)$ (in fact they have rate approaching $\Omega(\varepsilon)$ as the alphabet size grows) — details on this construction appear also in [10, Section 10.7].

It is also an interesting question whether a code construction similar to Theorem 5 can be obtained in time which is polynomial in both N and $1/\varepsilon$.

Acknowledgments

I would like to thank Piotr Indyk, Dan Spielman and Madhu Sudan for useful discussions.

References

1. N. Alon. Personal Communication, October 1999.
2. N. Alon, J. Bruck, J. Naor, M. Naor and R. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. on Information Theory*, 38 (1992), pp. 509-516.
3. N. Alon, O. Goldreich, J. Håstad and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3 (1992), pp. 289-304.

4. V. M. Blinovsky. Bounds for codes in the case of list decoding of finite volume. *Prob. Information Transmission*, **22** (1), pp. 11-25 (in Russian), 1986; pp. 7-19 (in English), 1986.
5. P. Elias. Coding for two noisy channels. *Information Theory*, Third London Symposium, September 1955, Buttersworth's Scientific Publications, pp. 61-76.
6. P. Elias. List decoding for noisy channels. *Wescon Convention Record*, Part 2, Institute of Radio Engineers (now IEEE), pp. 94-104, 1957.
7. P. Elias. Error-correcting codes for list decoding. *IEEE Trans. Info. Theory*, **37** (1), pp. 5-12, 1991.
8. G. D. Forney. *Concatenated Codes*. MIT Press, Cambridge, MA, 1996.
9. A. Gal, S. Halevi, R. J. Lipton and E. Petrank. Computing from partial solutions. *Proc. of 14th Annual IEEE Conference on Computation Complexity*, pp. 34-45, 1999.
10. V. Guruswami. *List Decoding of Error-Correcting Codes*. Ph.D thesis, Massachusetts Institute of Technology, August 2001.
11. V. Guruswami, J. Håstad, M. Sudan and D. Zuckerman. Combinatorial Bounds for List Decoding. *Proceedings of the 38th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, October 2000.
12. V. Guruswami and P. Indyk. Untitled Manuscript, April 2001.
13. V. Guruswami and M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45 (1999), pp. 1757-1767.
14. V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. *Proc. of 32nd ACM Symposium on the Theory of Computing (STOC)*, May 2000, pp. 181-190.
15. V. Guruswami and M. Sudan. The Johnson Bound: Revisited and Improved. *Submitted for publication*, December 2000.
16. R. R. Nielsen and T. Hϕholdt. Decoding Reed-Solomon codes beyond half the minimum distance. *Proceedings of the International conference on Coding Theory and Cryptography*, 1998.
17. R. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, **46**(1), pp. 246-257, January 2000.
18. J. H. van Lint. *Introduction to Coding Theory*. Graduate Texts in Mathematics **86**, (Third Edition) Springer-Verlag, Berlin, 1999.
19. J. M. Wozencraft. List Decoding. *Quarterly Progress Report*, Research Laboratory of Electronics, MIT, Vol. 48 (1958), pp. 90-95.
20. V. V. Zyablov and M. S. Pinsker. List cascade decoding. In *Prob. Information Transmission*, **17** (4), pp. 29-34 (in Russian), 1981; pp. 236-240 (in English), 1982.

Verification of a Leader Election Algorithm in Timed Asynchronous Systems

Neeraj Jaggi and K. Gopinath

Indian Institute of Science
gopi@csa.iisc.ernet.in

Abstract. The Timed Asynchronous System (TAS) model[3] has less stringent assumptions than the synchronous model but is still strong enough to serve as a foundation for the construction of dependable applications. In this paper, we verify the correctness of some basic distributed services in TAS. First, TAS is modelled and then some important properties of two basic services, FADS (Fail Aware Datagram Service) and HALL (Highly Available Local Leader Election Service), are formally verified. The PVS theorem prover is used for modelling and verification of the algorithms.
During the process of verification, some of the assumptions in the model that were not explicitly noted in the literature came to light. In addition, due to the insight gained in the process of verification, the ability to extend the validity of some of the properties in the face of additional failures in the system became clear through appropriate modifications of these assumptions.

1 Introduction

Distributed systems can be broadly characterized as synchronous, partially synchronous or asynchronous[2]. Synchronous systems involve strict timing constraints and bounded failure frequency assumptions while asynchronous systems, being a time-free model, do not involve any timing bounds. Asynchronous systems allow processors to crash and messages to be dropped or delivered late (performance failures). As strictly asynchronous systems lack control over the scheduling of events in the system. essential services like consensus, election or membership have been shown to be not implementable. A partially synchronous model is timing based with some restrictions on the relative timing of events; for example, it may assume almost-synchronized clocks, approximate bounds on process step time or message delivery time.

The Timed Asynchronous System (TAS) model[3], a timed model, has more complex assumptions than the synchronous model but which are easier to achieve in practical systems. They are, however, still strong enough to serve as a foundation for the construction of dependable applications. For example, the hardware clock drift rate is assumed to be bounded but the clocks of different processors are not assumed to be synchronized. In TAS:

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 207–218, 2001.
© Springer-Verlag Berlin Heidelberg 2001

1. All services are timed: A specification of a service prescribes a time bound within which state transitions are expected to occur.
2. Interprocess communication is unreliable: Messages can suffer omission failure (message dropped) and/or performance failure (message delivered late).
3. Processes have crash/performance failure semantics.
4. Processes have access to hardware clocks with bounded drift.
5. Frequency of communication and process failures is unbounded.

This timed model consists of finite set of processes that communicate via an unreliable datagram service. Processes have access to local hardware clocks. The datagram service provides for the send/receive primitives. A one-way time-out delay δ is chosen such that messages are likely to be delivered within δ real time units. Process management service ensures that a process has a maximum scheduling delay of σ real time units. Some other properties of timed model include communication by time and *conditional timeliness*. Conditional timeliness requires guaranteed progress within a bounded amount of time only when a set of processes form a stable partition, as against conventional timeliness that requires progress to be guaranteed within a bounded time unconditionally.

To guarantee liveness, progress or in real-time applications, it is often necessary to tag a message as fast or slow on delivery *a posteriori*. For example, in a leader election, extremely delayed messages may need to be ignored for progress[1]. They have to be ignored for correctness if they are from earlier rounds. Also, in security protocols, to avoid replay attacks.

The Fail Aware Datagram Service (FADS)[4] calculates an upper bound on the transmission delay of a message (Δ) and classifies all messages with transmission delay greater than Δ as *slow* and others as *fast*. A critical property that has to be shown for FADS is that a message that is fast may be delivered as slow but a slow message *must not* be delivered as fast. Various other services can be built upon fail awareness as shown in Figure 1.

Several protocols in TAS, e.g. HALL (highly available leader election service)[7], use a *leasing* or *locking mechanism* to communicate by passage of time. For example, during a "lease", there can be a promise not to change a value (value "locked"). The lease expires with or without communication; we only need some loose clock synchronization and some estimate on the delays to be able to make progress. We also require only one fail-aware datagram message instead of a round-trip message pair. Communication takes place between processes, even when the network is overloaded or partitioned. This mechanism is particularly useful for processes that switch a system to a safe mode.

Initial measurements on actual systems[3] give evidence that the timed model adequately describes current distributed systems such as a network of workstations. Also essential services, like leader election, are implementable[7]. Fail awareness can be applied to partitionable systems where communication is not certain due to network failures or excessive performance failures. Servers in each

[1] Note the lack of "progress" in the US presidential elections in Nov 2000 because of delayed ("slow") votes!

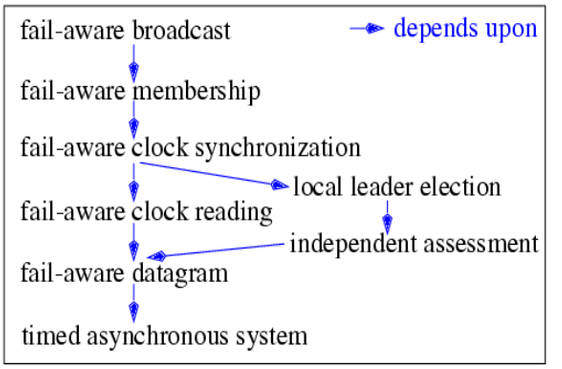


Fig. 1. Hierarchy of fail-aware services (from [3])

logical partition are allowed to make progress independent of servers in other partitions.

In this paper, we verify the correctness of some basic distributed services in TAS. First, TAS is modelled and then some important properties of two basic services, FADS (Fail Aware Datagram Service) and HALL (Highly Available Local Leader Election Service), are formally verified. The PVS theorem prover[6] is used for modelling and verification of the algorithms.

Section 2 briefly discusses the Timed Asynchronous System, its properties and advantages. Section 3 discusses two services implemented over TAS and Section 4 discusses the modeling and verification aspects involved using PVS.

2 Two Distributed Services in TAS

2.1 Fail Aware Datagram Service

In many practical situations in distributed systems, there is a need to reject old or stale messages. For instance, in leader election, a process needs to reject any message not belonging to the current round. Also to use the *leasing* mechanism of communication by time, a process has to be able to determine an upper bound on the one-way transmission delay of messages. FADS (Fail Aware Datagram Service) [1][4] calculates an upper bound for each message it delivers, and delivers the message as either *fast* or *slow*. FADS can be used in partitionable systems to provide "clean partitions", *i.e.* even when slow messages from other partitions arrive, higher level protocols see non-overlapping "logical partitions" [7].

A process computes for each message m it receives an upper bound on the transmission delay of m by measuring on its local clock the duration of an asynchronous datagram round trip which includes m. If the calculated upper bound is less than Δ, it is delivered as fast else as slow, where Δ is chosen as given below. Since timestamps of earlier messages are required to calculate the upper bound, it is required that only after two processes are connected

for some bounded time γ, all timely messages between them are delivered as fast. (A timely message has a transmission delay smaller than δ). The upper bound is defined as $ub_n(m) = (D - A) * (1 + \rho) - \delta_{min} - (C - B) * (1 - \rho)$ where δ_{min} is the minimum transmission delay of a message (can be set to 0). $fa_deliver_p^t(m, q, flag)$ denotes that message m from q is delivered to p at time t. The flag denotes whether the message is fast or slow. Two processes are said to be *connected* iff they are not crashed and all messages sent between them in the given time interval are timely. The FADS has the following properties in addition to the datagram service provided by the TAS model [3]:

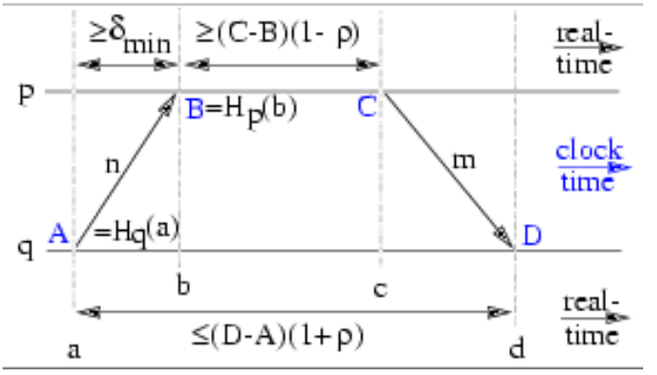


Fig. 2. Transmission delay td(m) = d-c is bounded by calculating an upper bound for the length of [a,d] and lower bounds for [a,b] and [b,c] (from [4])

1. Fail Awareness: $\forall p, q, m, t : fa_deliver_p^t(m, q, fast) \Rightarrow \exists s \leq t - \Delta : send_q^s(m, p) \vee broadcast_q^s(m)$
2. Timeliness:$\forall p, q, m, t : connected(p, q, t-\gamma, t+\Delta) \wedge (send_q^t(m, p) \vee broadcast_q^t(m)) \Rightarrow \exists s \leq t + \Delta : fa_deliver_p^s(m, q, fast)$.

Δ is chosen so that the calculated upper bound for any timely message $< \Delta$ i.e. $td(m) \leq ub_n(m) \leq \Delta$. Also a lower bound for γ is derived. Each process keeps arrays to store timestamps of messages sent or received from other processes. Each process broadcasts every τ clock time units so that all connected processes can update their timestamps for this process. Also it is interesting to note that fail-awareness sometimes allows a timely message to be delivered slow but a message with $td > \Delta$ is never allowed to be delivered fast! Measurements [4] have shown that by choosing τ, Δ, δ appropriately the calculated upper bound is quite close to the actual transmission delay and that the above mentioned properties hold. They are verified using PVS (see below).

2.2 Local Leader Election Service

In partitionable systems, one wants progress to be made in all partitions to increase the availability of services. But excessive performance failures make it impossible to elect a leader in all partitions. Hence the highly available leader election (HALL) problem[7] only requires that a leader is elected within a bounded amount of time in a stable partition

The unstable partitions are split into "logical partitions". A logical partition is a set of processes such that the local leader can communicate in a timely manner with all processes in the logical partition. The election of a local leader has to be supported by all processes connected to it. HALL creates logical partition so that

- Logical partitions never overlap
- Stable partitions are subset of logical partitions
- Two local leaders are always in two separate partitions
- Processes in one partition are not connected to the local leader of any other logical partition

HALL election service uses an independent assessment protocol to approximate the set of processes in a stable partition. The leadership is also renewed in a round-based fashion. A leasing mechanism to communicate by time is used to ensure that a process is at a time in at most one logical partition.

Process p is $\Delta - disconnected$ from q in $[s,t]$ iff all messages p receives in $[s,t]$ from q have a $td > \Delta$ real-time units. A $stablePartition\ SP$ is a Δ-partition that is defined as: $\Delta - partition(SP,s,t) \equiv (SP \neq \Phi) \wedge (\forall p,q \in SP : connected(p,q,s,t)) \wedge (\forall p \in SP, \forall r \in P - SP : \Delta - disconnected(p,r,s,t))$

The goal of the HALL service is to elect one leader in each stable partition. $Leader_p(t)$ denotes if process p is leader at time t. $supports^t(p,q)$ denotes that p supports q's election at time t. The HALL election service satisfies the following properties:

1. (T) Timeliness: a local leader is elected after at most k time units: $\forall SP \subseteq P, \forall t : stablePartition(SP, t-k, t) \Rightarrow \exists p \in SP, \exists s \in [t-k,t] : Leader_p(s)$
2. (SO) Support atmost one: a process cannot support election of 2 different leaders at same time: $\forall t, \forall p,q,r \in P : supports^t(p,q) \wedge supports^t(p,r) \Rightarrow q = r$
3. (BI) Bounded inconsistency: a process can be connected to 2 leaders simultaneously for at most k time units: $\forall p,q,t : connected(p,q,t-k,t) \wedge Leader^t(p) \Rightarrow supports^t(q,p)$
4. (LS) Leader self support: a leader supports self: $\forall p,t : Leader^t(p) \Rightarrow supports^t(p,p)$

SUPPORTS, the reflexive, symmetric and transitive closure of supports predicate, creates logical partitions. Each process uses a leasing (or locking) mechanism to communicate by time in supporting the leader. The protocol involves various parameters like

- $aliveSet_p$: set of processes from which p has received a fast message in the last *expires* clock time units

– $expirationTime$: a leader has to renew its leadership within this time
– $lockTime$: the time a process p supports another process q

The bounds for the constants k, $expires$, $lockTime$ are derived [7] so that the above four properties hold. Also the fact these four properties solve the local leader election problem in partitionable systems is verified.

3 Modeling and Verification

TAS is modeled in PVS using its specification language. Theories like reals, relations, `finite_sets` etc already present in PVS are imported and the related *lemmas* given in `prelude.pvs` are reused. For example, the theory `relations` is used to model the equivalence relation SUPPORTS while theory `finite_sets_def` is used to specify logical partitions. A modular approach, allowed in PVS through the import of theories, is followed to model the complete system.

The assumptions which the FADS and the HALL make are initially modeled as *axioms*. All the properties are formally verified using these *axioms* and the PVS prover commands and strategies[6]. Then the actual protocols for the FADS and HALL are modeled and the *axioms* assumed by the above services are proved to be correct.

3.1 Modelling Time Dependent Variables

In the leader election protocol, various variables e.g. $lastRequest_p$, flags e.g. $lockflag_p$ and sets e.g. $aliveSet_p$, $replySet_p$, $supportSet_p$, corresponding to each of the processes vary with time. For instance, the set $replySet_p$ of a process p is modified whenever a reply from any other process arrives supporting p's election. Also $lastRequest_p$ is modified whenever p broadcasts an election message. $lockflag_q$ is modified when q locks to/supports p (in which case it is set to true) or if $lockTime$ clock time units have expired after q locked to some process p (in which case it is set to false).

But these variables or sets are reinitialised before the next round starts. Also, *lockflag* is sticky once true within a round. Thus we can remove their time dependencies by taking snapshots at the beginning and end of rounds. Since the properties needed to be verified hold true at the end of a particular round, we need not look at the intermediate values of those variables, flags and sets.

3.2 Modelling Clocks Along with Approximations

Most of the parameters involved in TAS are reals. But they are also constrained e.g. ρ lies between 0 and 1, which is captured through subtyping.

```
x: VAR real
rhotype: NONEMPTY_TYPE = {x |x>0 AND x<1}
rho: rhotype
```

Also "realtime" and "clocktime" are defined to be two separate types. The hardware clock H_p maps realtime to the clocktime for process p. Processes are of type natural number and message is a non-empty type.

```
clocktime: TYPE = {x | x >=0}
realtime : TYPE = {x | x >=0}
proc: TYPE = nat
H(p: proc)(rt: realtime): clocktime
msg: NONEMPTY_TYPE
```

While trying to prove some *lemmas* and *theorems*, some implicit assumptions become apparent. They have been modeled as assumptions in PVS. For example, `OnetoOne` axiom states that the clocktime given by the hardware clock of any process corresponding to a particular realtime is unique. This is needed to model the strictly monotonous nature of the hardware clocks.

```
OnetoOne: AXIOM (FORALL (p: proc, s,t: rt): H(p)(s) = H(p)(t) IFF s = t)
```

ρ^2 is equated to zero only in the derivation of upper/lower bounds for various parameters. Hence the following axiom is needed. Since PVS uses only linear decision procedures, no inconsistency or derivation of $\rho = 0$ is possible.

```
RhoApprox: AXIOM rho*rho = 0
```

Since ρ^2 is close to zero, $(1+\rho)^{-1} \simeq (1-\rho)$ and $(1-\rho)^{-1} \simeq (1+\rho)$ where ρ is the maximum clock drift. To handle such approximations, another axiom is introduced in addition to the usual axiom for the correctness of the clock. `correctH1` can easily be mathematically verified using the above approximations and `correctH` axiom.

```
correctH(p:proc)(u:realtime): bool =
  FORALL (s:realtime,t:realtime | s < t AND t <= u):
  (1-rho)*(t-s) <= H(p)(t)-H(p)(s) AND (1+rho)*(t-s) >= H(p)(t)-H(p)(s))

correctH1(p:proc)(u:realtime): bool =
  (FORALL (s: realtime, t: realtime | s < t AND t <= u):
  t-s <= (H(p)(t)-H(p)(s))*(1+rho) AND t-s >= (H(p)(t)-H(p)(s))*(1-rho))

correctAX:AXIOM(FORALL(p:proc,u:realtime):correctH(p)(u)=>correctH1(p)(u))
```

3.3 FADS

"Fail Awareness" which is the underlying assumption in FADS ensures that a message with transmission delay $> \Delta$ is never delivered as "fast". Fail awareness itself is proved next from the datagram protocol specification.

```
AXIOM(FORALL(p:proc,q:proc,m:msg,t:rt): fa_deliver(t)(q)(m,p,fast) IMPLIES
  EXISTS (s:rt|s>=t-DELTA AND s<t): send(s)(p)(m,q) OR broadcast(s)(p)(m))

% a slow message will never be delivered fast
LEMMA1: LEMMA (FORALL (p:proc,q:proc,m:msg, s,t : rt):
  (t > DELTA + s AND (send(s)(p)(m,q) OR broadcast(s)(p)(m)) AND
  (FORALL (u:rt|u > s AND u < t): NOT (send(u)(p)(m,q)
  OR broadcast(u)(p)(m)))) IMPLIES NOT fa_deliver(t)(q)(m,p,fast))
```

The upper bound on the one-way transmission delay (see Section 2.1) is calculated. From the underlying protocol, min-delay assumption and bounded clock drifts, one can prove that the delay is bounded by the calculated upper bound.

```
ub(A,B,C,D: ct): real = (D - A)*(1 + rho) - (C - B)*(1 - rho) - deltamin

UBound: LEMMA (FORALL(A,B,C:ct, m:msg, p,q:proc, t:rt):
  (Send(A,B)(C,m)(p,q) AND Receive(A,B)(C,t,m)(q,p) AND correctH(p)(t)
  AND correctH(q)(t)) IMPLIES td(q)(m) <= ub(A,B,C,H(q)(t)))

%fail awareness
FAILAWARE:LEMMA(FORALL(p,q:proc,m:msg,t:rt): fa_deliver(t)(q)(m,p,fast)=>
  EXISTS(s: rt|s>=t-DELTA AND s<t): send(s)(p)(m,q) OR broadcast(s)(p)(m))
```

By making sure that $\Delta >$ calculated upper bound and by using `UBound` lemma above, the fail awareness property is proved.

To prove the "Timeliness" requirement, the lower bound on γ and a *helper* axiom are used. The axiom states that each process broadcasts messages every τ clock time units and that the processes are connected. Non-linear arithmetic lemmas from `real_props` theory and approximations as listed earlier are also needed.

```
%helper: each timely process p bcasts at least every tau clock time units
AXIOM (FORALL(s,t:rt, p:proc | H(p)(t)-H(p)(s)>=tau): Timely(p)(s)(t) =>
  EXISTS (u:rt,m:msg,C:ct|u>=s AND u<t AND H(p)(u)=C): Broadcast(C,m)(p))

%lower bound for gamma
GAMM: AXIOM gamma >= tau*(1 + rho) + delta

% timeliness requirement
TI: LEMMA (FORALL(p,q:proc,m:msg,t:realtime |t >= gamma):
  (connected(p,q)(t-gamma,t+DELTA) AND send(t)(p)(m,q)) IMPLIES
  EXISTS (s: realtime | s <= t + DELTA): fa_deliver(s)(q)(m,p,fast))
```

The effort involved in proving fail awareness and timeliness from the protocol properties is much higher compared to proving LEMMA1 assuming fail awareness since the datagram level protocol involves timestamps and non-linear bounds for parameters γ, Δ etc.

3.4 HALL

The HALL election service requires two invariants to be satisfied. Invariant INV0 states that two processes connected for k real time intervals cannot both be leaders at the same time. INV1 disallows the situation where there are two leaders in a trio of processes .

```
% timeliness condition T:
AXIOM FORALL(SP:setof[proc],t:rt|t>k): stablePartition(SP)(t-k,t) =>
  EXISTS (p:proc,s:rt | LiesIn(s)(t-k,t)): SP(p) AND Leader(p)(s)
```

```
%support at most one SO:
AXIOM FORALL(t:rt,p,q,r:proc):supports(t)(r,p) AND supports(t)(r,q) => p=q
% bounded inconsistency BI:
AXIOM FORALL(p,q:proc,t:rt | t>k):
  (connected(p,q)(t-k,t) AND Leader(p)(t)) IMPLIES supports(t)(q,p)
% leader supports itself LS:
AXIOM FORALL(p:proc,t:rt): Leader(p)(t) => supports(t)(p,p)

INV0: LEMMA FORALL(p,q:proc,t:rt | t>k AND p/=q):
  connected(p,q)(t-k,t) IMPLIES NOT(Leader(p)(t) AND Leader(q)(t))
% invariant: p,q,r no two leaders can exist (after k time units)
INV1: LEMMA FORALL(p,q,r:proc,t:rt | t>k AND p/=r): (connected(p,q)(t-k,t)
  AND connected(q,r)(t-k,t) AND Delta-disconnected(p,r)(t-k,t)) IMPLIES
  NOT (Leader(p)(t) AND Leader(r)(t))
```

Logical partitions (`logPart`) are formed by the "supports" predicate. That logical partitions don't overlap is implicit from the fact that SUPPORTS is an equivalence relation. Lemma OVERLAP states the fact. Also lemma LPLDR states that two leaders cannot exist simultaneously in a logical partition. Lemma SPLDR states that in a stable partition there can be at most one leader after a bounded time (k real time units).

To prove that no two leaders can exist in a logical partition, we use the fact that two processes, p and q, are in the same logical partition iff there exists an undirected path from p to q. The predicates "undirected" and "directed" are defined for the "supports" relation.

```
%equivalence relation on supports gives SUPPORTS
AXIOM FORALL(p,q:proc,t:rt):supports(t)(p,q) => SUPPORTS(t)(p,q)
AXIOM FORALL(t:rt): equivalence?(SUPPORTS(t))
AXIOM FORALL(t:rt,p,q:proc):SUPPORTS(t)(p,q) IFF logPart(t)(p)=logPart(t)(q)

%there exists a directed path from a to b using supports predicate
directed(a:proc)(b:proc)(t:rt):INDUCTIVE bool= (supports(t)(a,b) OR
  (EXISTS (c:proc | c /= a): supports(t)(a,c) AND directed(c)(b)(t)))
%undirected path from a to b in supports predicate
undirected(a:proc)(b:proc)(t:rt):bool=(directed(a)(b)(t) OR directed(b)(a)(t)
  OR(EXISTS(s:proc|s/=a AND s/=b):(directed(a)(s)(t) AND directed(b)(s)(t))))

% p,q belong to same logical partition only if undirected path(p,q,t)
AXIOM FORALL(p,q:proc,t:rt | p/=q AND logPart(t)(p)=logPart(t)(q)):
  undirected(p)(q)(t)
% logical partitions don't overlap
OVERLAP:LEMMA FORALL(p,q,r:proc,t:rt|p/=qANDp/=r):(logPart(t)(p)=logPart(t)(q)
  AND logPart(t)(p)=logPart(t)(r)) => logPart(t)(q)=logPart(t)(r)

%there can be at most one leader in a logical partition
LPLDR: LEMMA FORALL(p,q:proc,t:rt | p/=q): (logPart(t)(p)=logPart(t)(q))
  IMPLIES NOT (Leader(p)(t) AND Leader(q)(t))
%after k time units there is at most one leader per stable partition
SPLDR: LEMMA FORALL(SP:setof[proc],t:rt|t>k):stablePartition(SP)(t-k,t)=>
  NOT EXISTS(p,q:proc|p/=q):SP(p) AND SP(q) AND Leader(p)(t) AND Leader(q)(t)
```

The HALL protocol uses a leasing mechanism to communicate by time in process q supporting p's election. The support is valid only for *lockTime* clock time units. Lemma LOCK states that a process cannot be locked to two processes at a time, which is the invariant used to prove that a process can support at most one process at a time. Leader self support is proved from the fact that a process declares itself leader only if its *aliveSet* and *supportSet* are the same. Lemma HA1 states that a logical partition is formed k time units after the formation of stable partition. Also a process cannot be connected to a local leader of some other logical partition for more than a bounded amount of time (k) which is stated by lemma HA3.

```
% leader until expiration time
AXIOM FORALL(p:proc,t:rt): Leader(p)(t) IFF
  (H(p)(t)<expirationTime(p) AND imLeader(p))
% supports predicate definition: q supports p
AXIOM FORALL(p,q:proc,t:rt): supports(t)(q,p) IFF (Leader(p)(t) AND
  member(q,supportSet(p)))
% a process can be locked to at most one process at a time
LOCK: LEMMA FORALL(p,q,r:proc,t:rt|r/=p): NOT (locked(t)(q,p)
  AND locked(t)(q,r))
% if q supports p, q is locked to p
SUPPLOCK: LEMMA FORALL (p,q:proc, t:rt | t>gamma+2*DELTA):
  supports(t)(q,p)=>locked(t)(q,p)

% support at most one SO
SO:AXIOM FORALL(t:rt,p,q,r:proc | t > gamma + 2*DELTA):
  supports(t)(r,p) AND supports(t)(r,q) IMPLIES p = q
% leader supports itself
LS: AXIOM FORALL(p:proc,t:rt|t>gamma+2*DELTA):Leader(p)(t)=>supports(t)(p,p)
% Within k time units after a stable partition is formed, a logPart formed
HA1: LEMMA FORALL (t:rt,SP:setof[proc],p:proc | t > k):
  (stablePartition(SP)(t-k,t) AND SP(p) AND Leader(p)(t)) IMPLIES
  FORALL (q:proc |SP(q)): logPart(t)(q) = logPart(t)(p)
% processes in a partition not connected to local ldr of any other logPart
HA3:LEMMA FORALL(p,q:proc,t:rt| t>k): (Leader(q)(t) AND
  logPart(t)(p) /= logPart(t)(q)) IMPLIES NOT connected(p,q)(t-k,t)
```

The proof of lemmas INV0, INV1, SPLDR, LPLDR etc. from the axioms SO, LS etc. is quite straight forward. But to prove that the underlying protocol satisfies these axioms is more difficult. For example, consider trying to prove the axiom SO (support atmost once). After a stable partition is formed, process p broadcasts election request message to all the processes in its *aliveset* as soon as its own id becomes less than that of all processes in its aliveset. Process q, on receiving the election request, locks to process p if its *lockflag* is not set and p's id is less than that of all processes in q's *aliveset*. In this case q sets its *lockflag* to true for *expires* clock time units and sends a supportive reply to p. If p receives supportive reply from q, p adds it to its *supportset*. Process q supports p if and only if q belongs to p's supportset. Process q cannot lock to two processes simultaneously (lemma LOCK). Hence it suffices to prove that q supports p implies q is locked to p (lemma SUPPLOCK) to prove axiom SO. For

this to be true, process q's *lockflag* has to be set at the time process p checks for q's support.

One approach to proving lemma SUPPLOCK is to start with process p broadcasting an election request message. But with this approach, we need to able to prove that if *lockflag* is false and some conditions hold then *lockflag* is set to true which is quite difficult without temporal logic. A more suitable approach is to start with q supporting p. Hence p is the leader and q lies in the same partition as p. Also, q has received p's election request message and locked to p and *expirationtime* being less than *lockTime* clock time units proves the necessary property. Thus the dynamic changing aspect of *lockflag* is taken care of appropriately.

3.5 What Has Been Learned?

An interesting behavior of the system, which is not obvious but becomes apparent in the verification process, is that properties like "LS" and "SO" are valid only after $\gamma + 2 * \Delta$ real time units of the formation of the stable partition. This is so because it takes at least this much time for a process to declare itself leader. This also follows from the bounded inconsistency property but is not explicit.

Using PVS has helped us to uncover differing assumptions in the literature: for example, while FADS does not assume stable partitions, HALL assumes it. A casual reader might think that neither assumes it. The bounds derived for HALL assume the number of untimely messages as 0. We can generalize the results by giving *connected* one more parameter F (the maximum number of failures), using Δ-F partitions and deriving new bound for k in HALL (<= F * old k).

The number of steps involved in the process of verification is (> 150) with the size of the generated proof file being (> 1000 lines). Non-linearity of bounds for various parameters and case splits add to the effort involved (around 1000 lines of PVS code).

4 Related Work

In [5], a round-based synchronous algorithm modeled as a functional program was first transformed into an untimed system and then into a time-triggered system, in which bounds for various parameters were derived and the system was verified. It was also shown that the global states of the time-triggered system and the untimed system had a one-to-one correspondence. But the system we are considering is an event-based system where there is no bound on the transmission delays of messages. Also the notion of round in our case is different in different cases because of the complexity of the underlying protocol. For instance, the leader broadcasts election messages every election period, while a process locks to a leader for locktime interval and renewal of leadership takes place with a different time period. The assumptions in [5] include bounded drift and clock synchronization. Also the synchronous nature of the algorithm and the presence of rounds allows the implementation of the system as time triggered system

which simplifies its verification. In our case we deal with distributed algorithms in a partially synchronous system with a variable notion of a round which makes the modelling and verification more difficult. However, the calculation of bounds for the parameters in the algorithm is carried out in our case in a similar way. Faults in [5] were modeled by perturbation in functions. In our case the concept of Δ-Partition limits the number of message drops.

5 Conclusion

We believe that Timed Asynchronous Systems provide a better framework for distributed systems with its fail awareness and conditional timeliness guarantees. It solves the need for rejection of old messages and allows for essential services to be implemented even in partitionable systems.

We have studied Timed Asynchronous Systems and two of its services (fail awareness and local leader election) from a verification perspective. We were able to formally verify the properties of interest in this system (the PVS source files can be accessed at http://drona.csa.iisc.ernet.in/ gopi/fsttcs2001/). Approximations and time dependencies of variables have to be handled carefully. Verification of the system was useful and brought out many issues which were not quite obvious from the specification.

Acknowledgements

Thanks to the anonymous referees for their valuable comments and John Rushby and N. Shankar (SRI) for giving one of us (KG) the opportunity to study the problem while visiting SRI May-Jun'2000.

References

1. Christof Fetzer, "Fail Awareness in Timed Asynchronous Systems," *PhD thesis, CSE Dept, University of California, San Diego*, 1997.
2. Nancy A. Lynch, "Distributed Algorithms," Morgan Kaufmann, 1996.
3. Flaviu Cristian, Christof Fetzer, "The Timed Asynchronous Distributed System Model," *IEEE Transaction on Parallel and Distributed Systems, vol. 10, no. 6, pp. 642-657*, June 1999.
4. Flaviu Cristian, Christof Fetzer, "A Fail Aware Datagram Service," *2^{nd} annual workshop on Parallel and Distributed Systems*, April 5, 1997, *Geneva, Switzerland.*
5. John Rushby, "Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms," *IEEE Transactions on Software Engineering, vol. 25, no. 4*, September 1999.
6. Natarajan Shankar, Sam Owre, John Rushby, D. W. J. Stringer Calvert, "PVS Prover Guide, Version 2.3," September 1999.
7. Flaviu Cristian, Christof Fetzer, "A Highly Available Local Leader Election Service," *IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 603-618*, September/October 1999.

Efficient Addition on Field Programmable Gate Arrays

Andreas Jakoby [1] and Christian Schindelhauer [2]

[1] Institut für Theoretische Informatik, Med. Universität zu Lübeck
jakoby@tcs.mu-luebeck.de
[2] Heinz Nixdorf Institute and Depart. of Mathematics and Computer Science, Univ. Paderborn,
schindel@uni-paderborn.de

Abstract. We investigate average efficient adders for grid-based environments related to current Field Programmable Gate Arrays (FPGAs) and VLSI-circuits. Motivated by current trends in FPGA hardware design we introduce a new computational model, called the λ-wired grid model. The parameter λ describes the degree of connectivity of the underlying hardware. This model covers among others two-dimensional cellular automata for $\lambda = 0$ and VLSI-circuits for $\lambda = 1$. To formalize input and output constraints of such circuits we use the notion of input and output schemas. It turns out that the worst case time and area complexity are highly dependent on the specific choice of I/O schemas. We prove that a set of regular schemas supports efficient algorithms for addition where time and area bounds match lower bounds of a broad class of I/O schemas.
We introduce new schemas for average efficient addition on FPGAs and show that addition can be done in expected time $O(\log\log n)$ for the standard VLSI model and in expected time $O(\sqrt{\log n})$ in the pure grid model. Furthermore, we investigate the rectangular area needed to perform addition with small error probability, called area with high probability. Finally, these results are generalized to the class of prefix functions.

1 Introduction

To investigate the average time behavior of parallel systems we introduced the notion of average-complexity for circuits [16]. It turns out that for many basic Boolean functions like addition, threshold, or comparison there are special circuits, that provide a substantial speedup in the average for a wide class of probability distributions. This analysis was extended to a broader class of functions, the prefix functions, in [17,12,13]. It is not known how Boolean circuits with optimal average case behavior can be used in practice. A straight-forward approach is to use these averagely efficient circuits in asynchronous VLSI-design. However, new problems arise when it comes to verification, efficient combination, and placement of such circuits. Note that the analysis of expected time and worst time of a given circuit is a computationally infeasible task [14]. Determining these parameters is a precondition to predict hazards or to calculate local clocks. Hence, it seems to be necessary to design new circuits reflecting the restrictions given by the VLSI-environment.
A promising computational device for the implementation of average efficient circuits are Field Programmable Gate Arrays (FPGAs). FPGAs evolved from programmable array logics which can be configured to compute primitive Boolean functions, e.g. by

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 219–231, 2001.
© Springer-Verlag Berlin Heidelberg 2001

defining clauses of a conjunctive normal form. By the time more and more computational power was given to the basic logical units, called *logic blocks*, while the degree of chip integration increased their number. Programmable Gate Arrays introduced in [6] (see also [9,10,11] and [7,20] for surveys) can be used as an alternative to traditional hardware design: To implement the functionality of special processors it is cheaper and faster to configure an FPGA-chip, e.g. for mobile telephones or hand-held devices, than to make an Application Specific Integrated Circuit (ASIC). So, FPGAs extraordinarily reduce integrated circuit manufacturing time and prototype costs. The FPGAs' universal layout allows the emulation of every specific hardware after appropriate configuration. The underlying concept is a *sea of gates*: a uniform array of logic blocks with a regular interconnection network.

FPGAs of the latest generation, e.g. [1,2,23,24], provide up to some 60k free programmable logic blocks, some 64k bit RAM, and a clock frequency of 100 MHz, which on first sight seems not much compared to the current generation of processors and memory chips. But FPGAs are inherently parallel and fully programmable (some are even reconfigurable during computation). Furthermore, the integration of FPGAs has not yet reached the same level as of ASICs.

FPGAs are used as co-processors for high speed designs by implementing a variety of compute-intensive arithmetic functions including FFT, discrete-cosine transforms (DCT), and interpolators [22]. Furthermore, they are useful for cryptographical and cryptoanalytical applications: In [3] it is suggested to use FPGAs for a brute force attack against the DES crypto-system. More computational power is only achievable by a substantial longer development period and with higher costs.

There are two basic approaches to design the interconnection of gates within an FPGAs: the routing and the mesh design. In the *routing design* logic blocks are arranged along the boundary of the chip and the interior is completely occupied by a routing network. In the *mesh design* logic blocks are positioned in a grid and each block is connected to its direct neighbors. There is a restricted number of long wires facilitating long range interconnections of logic blocks. The mesh design allows a greater number of logic blocks. On the other hand its interconnection structure is more restricted than in the routing design.

In the mesh design the interconnection between logic blocks vary widely between manufacturers and between manufacturers' chip series. Some offer connections to diagonal adjacent neighbors, arbitrary zigzag-lines, buses for cells on the same row or column, connections to the second next neighbor, connections to the forth next neighbor, and so forth. Mesh based FPGAs provide a subset of these connection structures while not every combination of these may be used in a certain configuration. Also there are differences in the computational power of the logic blocks, in the number of clocks, and in the distribution and amount of memory. For a general approach it is necessary to define a computational model which covers a wide range of different FPGA designs.

In this paper we introduce a computational model for FPGAs, the λ*-wired grid model* motivated by common FPGA design principles [7]. The parameter λ addresses the amount of available long range interconnections. For $\lambda = 0$ it corresponds to the model of two dimensional cellular automata. For $\lambda = 1$ it describes the theoretical model of VLSI-circuits as discussed in [21]. If λ is large enough it models FPGAs in the routing

design. Like in VLSI we investigate time and area as complexity measures in the λ-wired grid model.
The computational complexity of basic functions like sorting, addition, multiplication, and many more are well studied in the VLSI-circuit model [21]. One achievement was to find ultimate lower bounds for time and area based on the planarity of chips and the speed of light. But, matching upper bound results do not necessarily lead to practical VLSI-algorithms. To achieve practical solutions input and output compatibility has to be realized. Our approach to I/O compatibility is the notion of *input* and *output schemas* to determine the meaning of an input or output bit at a pin position at any time.
We focus on the time and area complexity of the addition of long binary numbers as well as on arbitrary confluent prefix functions as introduced in [17]. For a specific input and output schema Brent and Kung [4] show that addition can be implemented in time $O(\log n)$ and area $O(n)$. This refers to the area occupied by the gates and the wires. It implements the Ladner-Fisher carry-propagation network [18] for partially pipelined input, i.e. an input is partitioned into a sequence of parallel blocks of equal length. In this paper we show that this bound is best possible for a wide class of schemas, called compact schemas. These bounds are also optimal, if the input and output schema are not compact, but closely related and if λ, the degree of connectivity, is arbitrarily high.
We introduce new schemas for efficient addition on FPGAs and show that addition can be done in expected time $O(\log\log n)$ for the standard (1-wired grid) model and in expected time $O(\sqrt{\log n})$ in the pure grid model. Furthermore, we investigate the size of a rectangular area that correctly performs addition with high probability, called *area w.h.p.* Following the ideas in [19] area w.h.p. can help reducing the over-all area, e.g. many adders using this smaller area share one large area consuming worst-case adder, which is only necessary for a small fraction of the inputs.
This paper is structured as follows. In Section 2 we introduce the notion of I/O schemas and a computational model for FPGAs, the λ-wired grid model. In Section 3 we present efficient schemas and prove tight bounds for the computational complexity of addition of two binary numbers. In Section 4 we focus on algorithms for the average case.

2 The λ-Wired Grid Model

An algorithm on a FPGA occupies a rectangular area $A = m_1 \cdot m_2$ consisting of $m_1 \times m_2$ programmable gates $(g_{i,j})_{i\in[m_1],j\in[m_2]}$, called *logic blocks* and an *interconnection network* connecting each logic block to four neighbor gates. Inputs and outputs may only occur at the $2m_1+2m_2$ *frontiers* $(g_{i,0})_{i\in[m_1]}$, $(g_{i,m_2+1})_{i\in[m_1]}$, $(g_{0,j})_{j\in[m_2]}$, and $(g_{m_1+1,j})_{j\in[m_2]}$ ($[n]$ denotes the set $\{1,..,n\}$). For the interconnection structure we distinguish between the *grid*, where interconnections are allowed only between adjacent gates, and the *λ-wired connection network*, where cells have a constant number of wires connecting them with distant neighbors according to the configuration of an FPGA connection network. The input and output occurs only on a subset of the frontiers, called *pins*. We will address the input pins by $p_1, \ldots, p_r$. The computation of a circuit is done in *rounds* (e.g. synchronized by a local clock). In each round a gate receives the output of its neighbors and computes a pre-configured function depending on these outputs and its state.

To obtain a high performance these algorithms should be integrated into a data flow. Therefore, it is very important that input and output time behavior is compatible. Pins may be used in a serial or parallel way to interchange input and output data. In general serial I/O helps to reduce the area needed for computation, while parallel I/O reduces the time needed for the whole computations. It is well known that a combination of these communication patterns can provide both qualities [4]. Our formalism for I/O compatibility is called *input* or *output schema*. We assume a unit time model for the *cycle time*, i.e. at each time step a new bit may arrive at each pin.

Definition 1. *An* **input or output schema (I/O schema)** $P \subseteq [n] \times [r] \times \mathbb{N}$ *of a vector* $x_1, \ldots, x_n$ *for pins* $p_1, \ldots, p_r$ *is a set of triples* (i, j, t)*, that describes that input bit* x_i *is available at pin* p_j *at time point* t*. For a schema* P *and pin* p_j *we define the start time* T_s *and the final time* T_f *by* $T_s(P, j) := \min_{(i,j,t)\in P} t$, $T_s(P) := \min_{(i,j,t)\in P} t$, $T_f(P, j) := \max_{(i,j,t)\in P} t$, *and* $T_f(P) := \max_{(i,j,t)\in P} t$.

The grid model, defined below, is the basic computational model for FPGAs in the mesh design (see Fig. 1). It is similar to two-dimensional cellular automata.

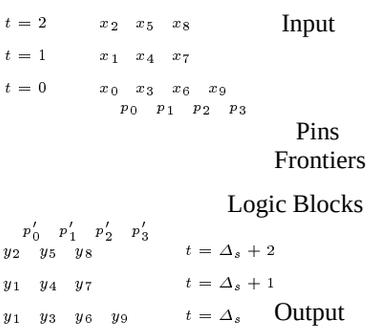


Fig. 1. The Grid-model

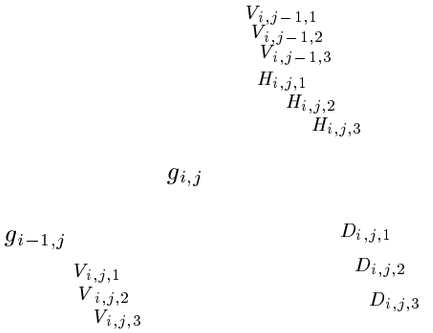


Fig. 2. The λ-wired grid model

Definition 2. *An algorithm in the* **grid model** *consists of an* $m_1 \times m_2$ *gate array* $(g_{i,j})_{i\in[m_1],j\in[m_2]}$ *and frontiers* $(g_{i,0})_{i\in[m_1]}$, $(g_{i,m_2+1})_{i\in[m_1]}$, $(g_{0,j})_{j\in[m_2]}$, *and* $(g_{m_1+1,j})_{j\in[m_2]}$ *which are grid like connected by* E
$\{\{g_{i-1,j}, g_{i,j}\} \mid j \in [m_2],\ i \in [m_1+1]\} \cup \{\{g_{i,j-1}, g_{i,j}\} \mid j \in [m_2+1],\ i \in [m_1]\}$.
In the configuration phase each logic block $g_{i,j}$ *can be configured to implement any function* $h_{i,j} : \Sigma^5 \to \Sigma^5$ *for a finite alphabet* Σ*. The computation phase starts at time* 0*. At every time step* t $(v_{i,j,t}, v^{\mathtt{n}}_{i,j,t}, v^{\mathtt{e}}_{i,j,t}, v^{\mathtt{s}}_{i,j,t}, v^{\mathtt{w}}_{i,j,t}) \in \Sigma^5$ *describes the state of a gate* $g_{i,j}$*. The internal state is given by* $v_{i,j,t}$*. Each of the other four values is transmitted to the corresponding neighbor. Unless gates are pins, for* $t = 0$ *all values are initialized with a special symbol* β*. This symbol* β *also describes the values of all non-pin frontiers. In each round* $t \geq 0$ *the following computation takes place*

$$(v_{i,j,t+1}, v^{\mathtt{n}}_{i,j,t+1}, v^{\mathtt{e}}_{i,j,t+1}, v^{\mathtt{s}}_{i,j,t+1}, v^{\mathtt{w}}_{i,j,t+1}) := h_{i,j}(v_{i,j,t}, v^{\mathtt{s}}_{i,j-1,t}, v^{\mathtt{w}}_{i+1,j,t}, v^{\mathtt{n}}_{i,j+1,t}, v^{\mathtt{e}}_{i-1,j,t}) .$$

The values of the pins are determined by the input string and an input schema. Unused positions of the I/O schema are indicated by the special symbol β.

FPGAs in the mesh design have an additional number of long wires for fast interconnections. The potential of high functionality of logic blocks results in larger area and slower time compared to a hardware layout of the same functionality. In FPGA design this is compensated by the number of wires which can be interconnected by *switches*. Such a switch connects the ends of four wires $\mathtt{n}, \mathtt{e}, \mathtt{s}, \mathtt{w}$ named by the cardinal directions. It features the following three *configurations*, which describe all three possible two-by-two connections, i.e. $\{\{\mathtt{n},\mathtt{s}\},\{\mathtt{e},\mathtt{w}\}\}$, $\{\{\mathtt{n},\mathtt{e}\},\{\mathtt{s},\mathtt{w}\}\}$, and $\{\{\mathtt{n},\mathtt{w}\},\{\mathtt{s},\mathtt{e}\}\}$. To simplify our model we assume bidirectional communication along the wires.

Definition 3. *For a switch or logic block* p *let* p^d *be its* **port** *to the cardinal direction* $d \in \{\mathtt{n},\mathtt{e},\mathtt{s},\mathtt{w}\}$. *For a port* p^d *let* $v(p^d,t) \in \Sigma$ *be the input value received by* p *from direction* d *at time* t *and* $w(p^d,t) \in \Sigma$ *the corresponding output value. If two ports* p^d *and* $q^{d'}$ *are connected we have* $v(p^d,t) = w(q^{d'},t)$ *and* $w(p^d,t) = v(q^{d'},t)$.

There is no common design principle in practice for the use of switches. For an overview see [7]. It turns out that many mesh based FPGA designs support the λ-wired connection model which is defined as follows:

Definition 4. *The* λ*-wired connection network connects logic blocks* $g_{i,j}$ *for* $i \in [m_1], j \in [m_2], d \in \{\mathtt{n},\mathtt{e},\mathtt{s},\mathtt{w}\}$ *with the switches* $H_{i,j,k}$, $V_{i,j,k}$ *and* $D_{i,j,k}$ *for* $i \in [m_1-1]$, $j \in [m_2-1]$, $k \in [\lambda]$. *The definition of the connections* E_λ *follows analogously to Fig. 2. For a configuration* C *of all switches, the resulting connection network* $E_\lambda(C)$ *describes a matching of all ports of logic blocks* $g_{i,j}$.

In practice there is a variety of connection networks for FPGAs which essentially provide similar features like the λ-connection network. The parameter λ varies for different FPGA designs, e.g. on the Xilinx XC4000E one can directly implement a 10-connection network and for the Xilinx XC4000X one can achieve $\lambda = 13$ (The switches used in the Xilinx architecture correspond to our model).
Note that the parameter λ also describes the relationship between size of logic blocks and area needed for wires, since for many hardware designs the area allocated for logic blocks is approximately as large as the area needed for wiring. We combine this connection network with the grid model:

Definition 5. *An algorithm in the* λ **-wired grid model** *consists of an* $m_1 \times m_2$ *gate array and a corresponding* λ*-connection network. In the configuration a function* $h_{i,j} : \Sigma^5 \to \Sigma$ *is assigned to the logic block* $g_{i,j}$ *and a configuration* C *of the network is chosen. The computation proceeds analogously to the computation in the grid-model except that in each round* $t \geq 0$ *the following computation takes place*

$$(q_{i,j,t+1}, w(g^{\mathtt{n}}_{i,j},t+1), w(g^{\mathtt{e}}_{i,j},t+1), w(g^{\mathtt{s}}_{i,j},t+1), w(g^{\mathtt{w}}_{i,j},t+1)) := h_{i,j}(q_{i,j,t}, v(g^{\mathtt{n}}_{i,j},t), v(g^{\mathtt{e}}_{i,j},t), v(g^{\mathtt{s}}_{i,j},t), v(g^{\mathtt{w}}_{i,j},t))) .$$

For $\lambda = 0$ the λ-wired grid model is equivalent to the grid model. An algorithm in the λ-wired grid model computes a function f w.r.t. I/O schemas P_I and P_O if for every input x given in accordance with P_I it outputs $f(x)$ using schema P_O.

Unlike for machine models like Turing machines or directed acyclic circuits it is not clear which notion of time for computation is most important for FPGAs. We use the following notions: The **over-all time** is the number of all time steps needed to compute the function, i.e. the time difference between the first input appears and the last output is delivered. The **latency** Δ_s is the time difference between the first given input and the first valid output. The **follow-up time** Δ_f is the time difference between the last available input bit and the last output. Of course the standard measure is the over-all time. The follow-up time is the decisive measure for pipelines consisting of many sub-circuits. The latency gives a measure for the best possible speedup, e.g. if another algorithm A waits for the outputs of a computation and A's computation depends only on one input element. A's computation may start even before the preceding computation (with small latency) is finished. To be able to take advantage of such effects we assume that there are acknowledgement mechanisms as presented in [8,16].
Since we consider the (potentially long) wires and the mechanism controlling the I/O of an algorithm as a valuable resource we prefer schemas that use every pin in every step of a time interval. We call such a schema 1-compact, or simply *compact*. If only a fraction of $\frac{1}{c}$ of all input (or output) possibilities is used during the schema's term, we call the schema c-compact.

Definition 6. *An schema P with n elements using r pins is called c-***compact***, if $r(T_f(P) - T_s(P) + 1) \leq c\,n$. We call 1-compact schemas* **compact***. A schema P is called a* **one-shot** *schema, if every input bit x_i occurs only once in P.*

Such a rigid notion of input and output behavior is not suitable for average-efficient circuits where some outputs may occur earlier. We allow earlier outputs if it does not disturb the chronological order of outputs at a pin.

Definition 7. *A* **relaxed schema** $RP(P) \subseteq 2^{[n]\times[m]\times\mathbb{N}}$ *is defined by a schema P and a set of functions $f \in \mathcal{F}$ with $f : [m] \times \mathbb{N} \to \mathbb{N}$ strictly monotone increasing w.r.t. time, the second parameter.*

$$P' \in RP(P) \quad :\Longleftrightarrow \quad \exists f \in \mathcal{F} \,:\, P' = \{(i,j,f(j,t)) \mid (i,j,t) \in P\}\,.$$

A δ-relaxed schema $RP_\delta(P) \subseteq 2^{[n]\times[m]\times\mathbb{N}}$ is a relaxed schema where the set of functions $\mathcal{F}$ is restricted by $f(j,t) \leq t + \delta$, for all j, t.

There are situation where it is reasonable to allow an output schema to change the topological and chronological order at the output pins. We call these kind of schemas *general relaxed schemas* as a specific union of schemas (we restrict ourselves to reasonable schemas, where the position and time of the transmitted string can be easily determined).

3 Efficient Schemas for the Addition

3.1 Definitions

First of all, we assume that for given binary input numbers $a, b \in \{0,1\}^n$ bits a_i and b_i arrive at a logic block at the same time. In the next step this logic block computes $x_i \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}$, using the mapping $(0,0) \mapsto \texttt{del}, (0,1) \mapsto \texttt{pro}, (1,0) \mapsto$

$\texttt{pro}, (1,1) \mapsto \texttt{gen}$, called the $\{\texttt{pro}, \texttt{gen}, \texttt{del}\}$-*representation* of a and b. Then the standard parallel prefix operation for addition has to be computed on the vector x_i to determine the result of the addition. For this we define the operator $\otimes : \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^* \to \{\texttt{pro}, \texttt{gen}, \texttt{del}\}$ by $\otimes(\varepsilon) = \texttt{pro}$ for the empty string ε and for any string $w \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^*$ define $\otimes(w\ \texttt{pro}) := \otimes(w)$, $\otimes(w\ \texttt{gen}) = \texttt{gen}$, and $\otimes(w\ \texttt{del}) := \texttt{del}$. The standard parallel prefix operator on an input $x \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^n$ is defined by $\mathrm{PP}_\otimes(x) := y_1 \dots y_n$ where $y_i = \otimes(x_1 \dots x_i)$. The sequence $(y_i)_{i\in[n]}$ directly describes the sum $a + b$.
Given the precalculated vector $(x_i)_{i\in[n]}$ we investigate the time and area complexity of the schemas bit-parallel word-serial (or **WS** for short), bit-serial word-parallel (or **WP** for short), bit-serial snakelike word-parallel (or **SWP** for short), snakelike bit-parallel word-serial (or **SWS** for short), and offset bit-parallel word-serial (or **OWS** for short) (see Fig 3). For converting a given schema into another schema for example WS into WP see e.g. [5].

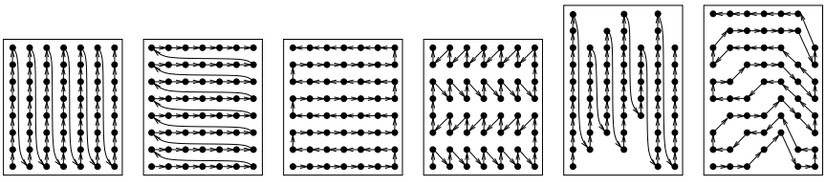


Fig. 3. Graphical description of the schemas: from left to right a) bit-parallel word-serial, b) bit-serial word-parallel, c) bit-serial snakelike word-parallel, d) snakelike bit-parallel word-serial, e) offset bit-parallel word-serial, and f) relaxed bit-serial snakelike word-parallel. In these diagrams the time is growing from the bottom to the top.

Definition 8. *Let $x_1, \dots, x_n$ be the input or output string. For r pins and $t = \lceil \frac{n}{r} \rceil$ we define the following schemas.*

$$r \times t\text{-WS} := \left\{ \left(i, \left\lceil \tfrac{i}{t} \right\rceil, 1 + (i-1) \bmod t\right) \mid i \in [n] \right\} \tag{1}$$

$$r \times t\text{-WP} := \left\{ \left(i, 1 + (i-1) \bmod r, \left\lceil \tfrac{i}{r} \right\rceil\right) \mid i \in [n] \right\} \tag{2}$$

$$\begin{aligned} r \times t\text{-SWP} := {} & \left\{ \left(i, 1 + (i-1) \bmod r, \left\lceil \tfrac{i}{r} \right\rceil\right) \mid i \in [n] \text{ and } \left\lceil \tfrac{i}{r} \right\rceil \text{ is odd} \right\} \cup \\ & \left\{ \left(i, r - (i-1) \bmod r, \left\lceil \tfrac{i}{r} \right\rceil\right) \mid i \in [n] \text{ and } \left\lceil \tfrac{i}{r} \right\rceil \text{ is even} \right\} \end{aligned} \tag{3}$$

$$\begin{aligned} (r,t,d)\text{-SWS} := {} & \left\{ \left(i, 1 + \left\lceil \tfrac{i}{d} - 1 \right\rceil \bmod r, (i-1) \bmod d + d \left\lceil \tfrac{i}{dr} \right\rceil\right) \mid i \in [n] \right. \\ & \left. \text{and } \left\lceil \tfrac{i}{dr} \right\rceil \text{ is odd} \right\} \quad \cup \quad \left\{ \left(i, 1 + \left\lceil \tfrac{i}{d} - 1 \right\rceil \bmod r, \right.\right. \\ & \left.\left. (i-1) \bmod d + \left\lceil \tfrac{i}{dr} \right\rceil \cdot d\right) \mid i \in [n] \text{ and } \left\lceil \tfrac{i}{dr} \right\rceil \text{ is even} \right\} \end{aligned} \tag{4}$$

r-OWS is a general relaxed schema with r pins with starting times $T_s(P,j)$ and lengths $t_j := T_f(P,j) - T_s(P,j) + 1$, where $n = \sum_j t_j$. The ith non-empty element at pin p_j corresponds to the element x_k for $k = \sum_{\nu<j} t_\nu + i$. (5)

3.2 Upper Bounds

Theorem 1. *In the grid model the addition of two n bit numbers with input schema P_I and output schema P_O can be computed in*
a) $A \in O(n), T \in O(r+t), \Delta_s, \Delta_f \in O(r+t)$ *for* $r \times t$ *-WS* P_I, P_O ,
b) $A \in O(r^2), T \in O(r+t), \Delta_s, \Delta_f \in O(r)$ *for* $r \times t$ *-SWP* P_I, P_O ,
c) $A \in O(dr+r^2), T \in O(d+r+t), \Delta_s, \Delta_f \in O(d+r)$ *for* (r,t,d) *-SWS* P_I, P_O ,
d) $A \in O(r \log t)$, $T \in O(r+t)$, $\Delta_s = 1$, $\Delta_f \in O(r+t)$ *for* $r \times t$ *-WS* P_I *and* r *-OWS* P_O .

Sketch of the algorithms: For all algorithms pins are located only in $(g_{0,j})_{[m_2]}$.
a) The algorithm for the WS I/O schema consists of three phases: In the first phase the algorithm stores the input of each pin p_i by using a FIFO-data structure Q_i . Furthermore, it applies $\otimes$ to all inputs received at p_i yielding $z_i = \otimes_{j=1}^{t} x_{(i-1)k+j}$. Then, it computes $\mathrm{PP}_\otimes(z_1 \dots z_r) =: k_1 \dots k_r$. Finally, it generates the output for each pin p_i by applying the prefix operator $\mathrm{PP}_\otimes$ with initial value k_{i-1} to the elements in Q_i .
d) We use a similar strategy for the OWS output schema. In first phase the algorithm counts the number c_i of leading `pro`s at each pin p_i . While this counting takes place no output is generated at p_i . If the algorithm receives a non-propagate symbol at p_i , it generates the output on-line at p_i determined by the rest of the input on p_i . As in the WS output schema, it computes the partial results z_i . In the second phase it computes $\mathrm{PP}_\otimes(z_1 \dots z_r) =: k_1 \dots k_r$. Finally, it generates c_{i+1} copies of k_i on pin p_i .
b) For the SWP I/O schema we pipeline the input through an $r \times 2r$ -field. For the first row we define the internal values $z_i^0 = x_i$, which appear according to the SWP schema. In the t -th row we calculate $z_i^t = z_{i-1}^{t-1} \otimes z_i^{t-1}$. Note that diagonal connections can be simulated by an appropriate encoding, e.g. increasing the alphabet Σ . In the last row of the field no values `pro` remain unless the input contains a `pro`-chain of length at least $2r$. If $z_i^{2r} = \texttt{pro}$ then the last non-`pro` value that was computed at this logic block $g_{i,2r}$ determines the output value y_i . Using standard techniques one can fold this field such that inputs and outputs are placed on the same side of the array.
c) The algorithm for the SWS I/O schema is a straight-forward combination of the algorithms for the SWP and the WS schema. ∎
For the λ -wired grid model we omit the WS and the SWS schema since they do not provide more efficient algorithms than the WP and the OWS schemas neither in the worst case nor in the average case.

Theorem 2. *For $\lambda > 0$ in the λ -wired model addition of two n bit numbers can be performed with input schema P_I and output schema P_O as follows:*
1. *if P_I and P_O are $r \times t$ -WP schemas then $A \in O(\min(n + \frac{n \log n}{\lambda},\ r \log r))$, $T \in O(t + \log r)$, and $\Delta_s, \Delta_f \in O(\log r)$.*
2. *if P_I is a $r \times t$ -WS and P_O is a r -OWS schema then $A \in O(r \log t + \frac{r \log r}{\lambda})$, $T \in O(t + \log r)$, $\Delta_s \in O(1)$, and $\Delta_f \in O(\log t)$.*

The algorithms for the λ -wired model corresponds to the grid-based algorithms except that we use a tree structure as described in [4] to compute the carries of parallelly received input elements (for the WP schema) or to compute $\mathrm{PP}_\otimes(z_1 \dots z_r)$ (for the OWS schema). The algorithm presented in [4] pipelines through a tree structure to

improve the area. For the OWS schema we use the carry computation tree only once to compute $\mathrm{PP}_{\otimes}(z_1 \ldots z_r)$. Hence, we can reduce the area needed to implement the tree structure by a factor of λ. This does not help for the WP schema since we cannot compress information processed in the pipeline. The following corollary improves the AT^2 bound of $O(n \log^2 n)$ shown in [4] for high connectivity parameter λ ($\operatorname{llog} n := \log \log n$).

Corollary 1. *In the $\log n / \operatorname{llog} n$-wired-model the addition of two n bit numbers with input schema $n/\log n \times \log n$-WS and output schema $n/\log n$-OWS needs area $A = O(n \operatorname{llog} n / \log n)$ and over-all time $T = O(\log n)$, thus $AT^2 = O(n \log n \operatorname{llog} n)$.*

3.3 Lower Bounds

Here, we investigate the question whether improved I/O schemas exist which enable more efficient adders than those presented in the previous section. The only restrictions are that either the input and output behaviors are similar or that both schemas are compact. The two following theorems show that indeed the presented schemas and algorithms are optimal w.r.t. time and area. Since both Theorems use similar arguments we present a proof sketch after Theorem 4.

Theorem 3. *In the grid model any algorithm adding two n bit numbers using r pins and schema duration t has an over-all time of at least $\frac{n}{r} + \sqrt{r}$ for any input schema and $t + r$ for any one-shot compact input schema. Furthermore, it has*
1) at least area $\Omega(\min(r^2,\ n)\ -\ \delta r)$ for any one-shot compact δ-relaxed input schema and relaxed output schema of the same type;
2) at least area $\Omega(\min(r^2,\ n))$ for any one-shot compact input schema and any one-shot compact output schema;
3) at least area $\Omega(n)$ for $\alpha \geq 1$ and $2\alpha t \leq \sqrt{n}$ for any one-shot compact input schema and one-shot α-compact output schema.

Theorem 4. *For the λ-wired grid model any algorithm adding two n bit numbers using r pins and schema duration t the over-all time is at least $\frac{n}{r} + \log r$ for any one-shot compact input schema. Furthermore, it has*
1) at least area $\Omega(\min(r\ \log r,\ n)\ -\ \delta r)$ for any one-shot compact δ-relaxed input schema and relaxed output schema of the same type;
2) at least area $\Omega(\min(r\ \log r,\ n))$ for any one-shot compact input schema and any one-shot compact output schema;
3) area $\Omega(n)$ for $\alpha \geq 1$ and $2\alpha t \leq \log n$ for any one-shot compact input schema and one-shot α-compact output schema.

Proof Sketch: 1.: Let P_I be a one-shot compact input schema and $P_O \in RP(P_I)$. Let

$$A := \{\, i \mid \exists j :\ (i,j,0) \in P_I \text{ and } |\{k < i \mid \exists j :\ (k,j,0) \in P\}| > r/2 - 1 \,\}$$
$$B := \{\, i \mid \exists j, \ell :\ (i,j,\ell) \in P_I \quad \text{and} \quad \ell \geq t/2 \,\}$$
$$B_j := \{\, i \mid \exists \ell :\ (i,j,\ell) \in P_I \,\} \cap B\ .$$

Note that $|A| \geq \frac{r}{2}$. For $i \in A$ define $C_i := \{\, j \notin B \mid \forall k \in \{j+1, .., i\} :\ k \notin B \,\}$. We choose $x_i := \texttt{pro}$ for all $i \in \overline{B}$ and we select for all $i \in B$ the value of $x_i \in \{\texttt{del}, \texttt{gen}\}$ such that the inputs in B has a high Kolmogorov complexity.

Note that each output y_i with $i \in A$ either depends on an input bit in B or on a pro-chain C_i containing at least $\frac{r}{2}$ input elements received by the algorithm at the beginning on different pins. Hence, each of these outputs is delayed by at least $\min\{\log(r/4), t/2\}$ steps, resp. by $\min\{r/4, t/2\}$ steps in the grid model. This implies that every output element at this pin is delayed by at least this number of steps. Therefore, also the output elements corresponding to B_i will be delayed and have to be stored. Because of the high Kolmogorov complexity of the input sequence addressed by B the algorithm needs a memory of size at least $r/2 \cdot \min\{\log(r/4), t/2\}$ for the λ-wired model, resp. $\min\{r/4, n/4\}$ for the grid model.
A δ-relaxation of the input schema w.r.t. a compact schema P reduces this bound by δr, because elements addressed by B can be delayed by δ steps in the input schema.
2. and *3.*: Now let P_I, P_O be one-shot compact schemas and $i_1 < i_2 < \cdots < i_r$ be the sequence of indices of the input which arrive at time step $t-1$. Define

$$A := \{\, i_1, \ldots, i_r \,\}, \quad B_1 := \{\, i \mid i_1 \le i \le i_{\lceil r/2 \rceil} \,\}, \quad B_2 := \{\, i \mid i_{\lceil r/2 \rceil + 1} \le i \le i_r \}$$

$$C_k := \{\, i \mid \exists j, \ell : (i, j, \ell) \in P_I \wedge \ell < k \,\}, \quad D_k := \begin{cases} B_1 \text{ if } |B_1 \cap C_k| \le |B_2 \cap C_k| \\ B_2 \text{ if } |B_1 \cap C_k| > |B_2 \cap C_k| \,. \end{cases}$$

For all $i \in D_k$ we choose $x_i = \texttt{pro}$ and for all $i \notin D_k$ we choose $x_i \in \{\texttt{del}, \texttt{gen}\}$ such that this sequence has a high Kolmogorov complexity. Note that the last position y_i in D_k can only be determined after $t + \log(r/2)$ steps. Assume that P_O is α-compact with $1 \le \alpha \le \frac{t + \log(r/2) - k}{t}$ for an appropriately chosen k. Then, any algorithm cannot generate the first output element before time step $t + \log(r/2) - \alpha \cdot t > k$. Hence, the algorithm has to withhold the output corresponding to C_k. Note that C_k addresses the input elements received within the first k steps. Note that at least half of these are not in D_k and possess high Kolmogorov complexity. This implies $A \in \Omega(\min(n,\ r \cdot k))$. By choosing $k = \log(r/2) + (1-\alpha)t$ the claim follows. ∎

4 Efficient on the Average

In this section we investigate average bounds for over-all time and follow-up time. Furthermore, we measure the *area necessary with high probability (area w.h.p)* defined as follows. Consider a partition of the over-all area of an algorithm into two rectangles R_0 and R_1 with area A_0 and A_1 where all pins are adjacent to R_0. The algorithm needs area A_0 w.h.p. if for a random input of length n the communication between R_0 and R_1 consists only of special symbols β with probability $1 - n^{-c}$ for some constant c.
Synchronizing output elements is an area-consuming task. For this reason, we use relaxed output schemas allowing results to be given out as soon as they are available. However, all of the here presented algorithms except the OWS schema produce compact (non-relaxed) output schemas w.h.p.

Lemma 1. *For any $c > 0$ the probability that the $\{\texttt{pro}, \texttt{gen}, \texttt{del}\}$-representation W of two uniformly chosen binary numbers $A, B \in \{0,1\}^n$ contains a contiguous propagate sequence of length $2(c+1) \cdot \log_3 n$ is bounded by n^{-c}.*

The algorithms used for averagely efficient addition are similar to the algorithms as presented in section 3 except that they generate outputs as soon as they are available.

Intuitively, Lemma 1 implies that for the algorithm for the SWP schema presented above it is sufficient w.h.p. to use a $r \times O(\log n)$ -field. For the OWS schema the counters c_i can be bounded by $O(\log n)$ w.h.p. For the relaxed WS and OWS output schema the algorithm has to resolve `pro` -chains $z_1 \dots z_n$, which are shorter than $O(\frac{\log n}{t})$ w.h.p.

Theorem 5. *In the grid model the addition of two n bit random bits with input schema P_I and output schema P_O can be computed within time $T = t + \Delta_f$ and:*

P_I	P_O	$E[\Delta_f]$	A *w.h.p.*
$r \times t$-*WS*	*rel.* $r \times t$-*WS*	$O(t + \frac{\log n}{t})$	$O(n)$
$r \times t$-*WS*	r-*OWS*	$O(\min(\log n, t + \frac{\log n}{t}))$	$O(r \min(\llog n, \log t))$
$r \times t$-*SWP*	*rel.* $r \times t$-*SWP*	$O(\min(\log n, r))$	$O(r \min(\log n, r, t))$
(r,t,d)-*SWS*	*rel* (r,t,d)-*SWS*	$O(d + \min(r, \frac{\log n}{d}))$	$O(r \min(d + r, \log n, t))$

It turns out that the algorithm addressed by the following Lemma is a basic building block for the design of average efficient adders for λ -wired grids.

Lemma 2. *There exists an $O(n \llog n)$ area and $O(\llog n)$ time bounded algorithm in the 1-wired grid model that computes the addition of two n bit random numbers w.h.p. if the input and output is given in parallel ($n \times 1$ -WP).*

Proof Sketch: We partition the input into blocks $U_i := x_{i\ell+1}, .., x_{(i+1)\ell}$ of length $\ell := c \log n$ for an appropriately chosen constant c and compute $\mathrm{PP}_\otimes(U_i U_{i+1})$ for each i using the algorithm of [4]. Let $y_{(i+1)\ell+1}, \dots, y_{(i+2)\cdot\ell}$ be the suffix of the result of this computation. Lemma 1 implies that $y_1, .., y_n$ is the correct result whp. ∎

Using the worst-case algorithms one can improve time and area by replacing all subroutines computing $\mathrm{PP}_\otimes$ by the algorithm of Lemma 2. To guaranty correctness if a propagate-chain of length $c \log n + 1$ occurs, the computation is delayed in such a case, and a worst-case algorithm takes over for the rest of the calculation.

Theorem 6. *In the λ -wired model adding two n bit random numbers with input schema P_I and output schema P_O can be computed in over-all time $T = t + \Delta_f$, where*

- *$E[\Delta_f] \in O(\min(\log n, t + \llog n))$, $A \in O(r(\min(\log t, \llog n) + \frac{\llog n - \log t}{\lambda}))$ whp for $r \times t$ -WS P_I and r -OWS P_O resp.*
- *$E[\Delta_f] \in O(\min(\log r, \llog n))$ and $A \in O(r \min(\log r, \llog n, t + \frac{t \llog n}{\lambda}))$ whp for $r \times t$ -WP P_I and rel. $r \times t$ -WP P_O .*

For a generalization, we observe that only the algorithm constructed for the OWS output schema uses a special property of the addition. All other bounds, presented here, can be generalized to a broader class of prefix functions. The worst case upper bounds presented in Section 3 holds for any prefix function. The average bounds presented in Section 5 hold for a sub-class of these functions, the so-called *confluent prefix functions*, introduced in [17]. Furthermore, the algorithms provide the same average complexity measures if the input probability distribution is generalized to *binomial approximable distributions* as dicussed in [13]. The lower bounds presented in Section 4 also apply for *diffluent prefix functions* introduced in [13]. Because of space limitation we omit the definitions of diffluent and confluent prefix functions.

As already noted, the algorithms for the offset schemas cannot be applied directly to general prefix functions. The following Theorem shows that there exists an alternative average efficient algorithm using this I/O schema for confluent prefix functions.

Theorem 7. *In the grid model the computation of confluent prefix function for a random input of length n with input $r \times t$ -WS input schema and r -offset output schema can be computed within area w.h.p. $O(r \min(\sqrt{\log n}, \sqrt{t}))$ and expected follow-up time $O(\min(\log n, t + \frac{\log n}{t}))$. In the λ -wired grid model the corresponding area bound w.h.p. is $O(r(\min(\sqrt{\log n}, \sqrt{t}) + \frac{\text{llog } n - \log t}{\lambda}))$ while the expected follow-up time is $O(\min(\log n, t + \text{llog } n))$.*

The key idea of the proof is to replace the counters of the corresponding adders by an area efficient data structure for storing intermediate values.

5 Conclusions

The results presented above can be used to optimize the average behavior of an FPGA if an appropriate schema is chosen. Table 1 summarizes the results of Theorem 5. In Table 1 we optimize the time implied by the output schema. If schemas have equal asymptotic expected time behavior we present the one with smaller area. The expected follow-up time of the relaxed SWS schema is minimized by choosing $d = \min(\sqrt{\log n},\ r,\ t)$. From Theorem 6 it follows for the λ -wired grid model there is no asymptotical difference in the expected over-all time $E[T] = t + E[\Delta_f]$ for the relaxed WP and OWS output schema. Only for $t \in o(\log n)$ and connectivity parameter $\lambda \in \omega(1)$ the OWS schema provides a more area efficient algorithm than the relaxed WP. For other parameters it seems that the relaxed WP schema should be preferred. Note that the corresponding algorithm produces a compact (non-relaxed) WP output schema w.h.p.

Table 1. Output schemas which allow expected time efficient algorithms for grids.

$r \in O(\sqrt{\log n})$	$r \in \omega(\sqrt{\log n}) \cap o(\frac{n}{\sqrt{\log n}})$	$r \in \Omega(\frac{n}{\sqrt{\log n}}) \cap o(n)$	$r \in \Omega(n)$
rel. SWP rel. SWS	rel. SWS	OWS	rel. WS, rel. SWP rel. SWS, OWS
$E[\Delta_f] = O(\frac{n}{t})$ $A \in O(r^2)$	$E[\Delta_f] \in O(\sqrt{\log n})$ A whp$\in O(r \min(\log n, r, t))$	$E[\Delta_f] \in O(\frac{\log n}{t})$ A whp$\in O(r \log t)$	$E[\Delta_f] \in O(\log n)$ $A \in O(n)$

References

1. Actel Corporation, *ProASIC $^{\text{TM}}$ 500K Family*, Product Spec., October 2000.
2. Atmel Corp.,*AT 40K FPGAs with FreeRAM $^{\text{TM}}$* , Rev. 0896B-01/99, Prod. Spec., Jan. 1999.
3. M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, M. Wiener, *Minimal key lengths for symmetric ciphers to provide adequate commercial security: A report by an ad hoc group of cryptographers and computer scientists*, 1996, www.bsa.org.
4. R. Brent, H.T. Kung, *A Regular Layout for Parallel Adders*, IEEE Transaction on Computers, C-31, 1982, 260-264.
5. W. P. Burleson, L. L. Scharf, *Input/Output Design for VLSI Array Architectures*, Proceedings VLSI'91, 8b2.1-10, 1991.

6. W. Carter, K. Duong, R. Freeman, H. Hsieh, J. Ja, J. Mahoney, L. Ngo, S. Sze, *A User Programmable Gate Array*, Proc. CICC'86, 1986, 233-235.
7. K. Compton, S. Hauck, *Configurable Computing: A Survey of Systems and Software*, Northwestern University, Dept. of ECE Technical Report, 1999.
8. I. David, R. Ginosar, M. Yoelli, *An Efficient Implementation of Boolean Functions and Finite State Machines as Self-Timed Circuits*, ACM SIGARCH, 1989, 91-104.
9. K. El-Ayat, *A CMOS Electronically Configurable Gate Array*, Proc. ISSCC, 1988, 76-77.
10. A. El Gamal, *An Architecture for Electronically Configurable Gate Arrays*, Proc. CICC'88, 1988, 15.4.1- 15.4.4.
11. H. Hsieh, K. Duong, J. Ja, R. Kanazawa, L. Ngo, L. Tinkey, W. Carter, and R. Freeman, *A Second Generation User Programmable Gate Array*, Proc. CICC'87, 1987, 515-521.
12. A. Jakoby, *Die Komplexität von Präfixfunktionen bezüglich ihres mittleren Zeitverhaltens*, PhD dissertation, University of Lübeck, 1998.
13. A. Jakoby, *The Average Time Complexity to Compute Prefix Functions in Processor Networks*, Proc. 16th STACS, 1999, 78-89.
14. A. Jakoby, C. Schindelhauer, *On the Complexity of Worst Case and Expected Time in a Circuit*, Proc. 13th STACS, 1996, 295-306.
15. A. Jakoby, R. Reischuk *Average Case Complexity of Unbounded Fanin Circuits*, Proc. 15th Conference on Computational Complexity (CCC), 1999, 170-185.
16. A. Jakoby, R. Reischuk, C. Schindelhauer, *Circuit Complexity: from the Worst Case to the Average Case*, Proc. 26th SToC, 1994, 58-67.
17. A. Jakoby, R. Reischuk, C. Schindelhauer, S. Weis *The Average Case Complexity of the Parallel Prefix Problem*, Proc. 21st ICALP, 1994, 593-604.
18. R. Ladner and M. Fischer, *Parallel prefix computation*, J. ACM, 27 (4), 1980,831–838.
19. J. Reif, *Probabilistic Parallel Prefix Computation*, Comp. Math. Applic. 26, 1993, 101-110.
20. R. Tessier, W. Burleson, *Reconfigurable Computing for Digital Signal Processing: A Survey*, to appear in Y. Hen Hu (ed) *Programmable Signal Processors*, Marcel Dekker Inc., 2001
21. J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.
22. J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, *Programmable Active Memories: Reconfigurable Systems Come of Age*, IEEE Trans. VLSI Systems 4 (1), 1996.
23. Xilinx Corp., *XC4000E and XC4000X Series Field Programmable Gate Arrays*, Prod. Spec., Version 1.6, 1999.
24. Xilinx Corp., *Virtex-II Platform FPGA Data Sheet (DS031)*, Prod. Spec., Ver. 1.5, 2001.

The Directed Minimum-Degree Spanning Tree Problem*

Radha Krishnan and Balaji Raghavachari

Computer Science Department, University of Texas at Dallas,
Richardson, TX 75083-0688, USA.
{Radha.Krishnan,Balaji.Raghavachari}@utdallas.edu

Abstract. Consider a *directed* graph $G = (V, E)$ with n vertices and a root vertex $r \in V$. The DMDST problem for G is one of constructing a spanning tree rooted at r, whose maximal degree is the smallest among all such spanning trees. The problem is known to be NP-hard. A quasi-polynomial time approximation algorithm for this problem is presented. The algorithm finds a spanning tree whose maximal degree is at most $O(\Delta^* + \log n)$ where, Δ^* is the degree of some optimal tree for the problem. The running time of the algorithm is shown to be $O(n^{O(\log n)})$. Experimental results are presented showing that the actual running time of the algorithm is much smaller in practice.

1 Introduction

The minimum-degree spanning tree (MDST) problem for an undirected graph $G = (V, E)$ is that of constructing a spanning tree of G, whose maximal degree is the smallest among all spanning trees of G. It is a generalization of the Hamiltonian Path problem and thus is also NP-hard. The problem can be defined on directed graphs as follows. Given a root vertex $r \in V$, find an incoming (or outgoing) spanning tree rooted at r, known as a branching, in which the maximal indegree (outdegree) of a vertex is minimized. We will refer to the directed version of the MDST problem as the DMDST problem. In the Steiner case, a set of terminals $D \subseteq V$ is also specified. The output tree must span the set D. However, it may contain any of the other vertices of G.

The general problem of computing low-degree trees is both fundamental and finds ready applicability, such as in noncritical broadcast and VLSI layout problems. It is also inherently appealing due to its seeming simplicity. Previous polynomial-time algorithms [2] for the DMDST problem find trees whose degree is at most $O(\Delta^* \log n)$, i.e., a *factor* of $\log n$ from the optimal degree Δ^*. On the other hand, the algorithm in [4] for *undirected* graphs finds a tree whose degree is at most $\Delta^* + 1$, i.e., an additive constant 1 away from the optimal. Our work tries to bridge this gap in performance of approximation algorithms for the undirected and directed versions of the MDST problem.

* Research supported by the National Science Foundation under grant CCR-9820902.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 232–243, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Our algorithm for the DMDST problem finds a tree whose maximal degree is at most $c\Delta^* + \lceil \log_c n \rceil = O(\Delta^* + \log n)$, for any constant $c > 1$. The approximation quality has therefore been improved from a multiplicative factor of $\log n$ to an additive term of $\log n$. The running time of the algorithm is shown to be quasi-polynomial, $O(n^{\log_c n + O(1)})$. However, it is conjectured that a better bound may be provable, and also that the running time may be much better in practice for this reason. We present experimental evidence to show that in practice the algorithm ran much faster than the theoretical bound obtained here. Also, the degree of the tree output is often very close to the optimal degree.

Previous work in undirected graphs. The first result on approximating a minimum-degree spanning tree was that of Fürer and Raghavachari [2]. They gave a polynomial-time approximation algorithm that returns a tree whose degree is at most $O(\Delta^* \log n)$. The first polynomial-time approximation algorithm for the Steiner version of the problem was provided by Agrawal, Klein and Ravi [1]. Their approximation ratio is $O(\log |D|)$, where D is the set of terminals. Ravi, Raghavachari and Klein [12] studied generalizations of the MDST problem and gave quasi-polynomial time approximation algorithms. Fürer and Raghavachari [4] improved their previous results and provided a new polynomial-time algorithm to approximate the MDST problem to within one of optimal. Their algorithm also extends to the Steiner version of the problem, but only works on undirected graphs. A survey of these and other minimum-degree problems has appeared in a book on approximation algorithms [9].

Ravi, Marathe, Ravi, Rosenkrantz and Hunt [11] proposed algorithms for computing low-weight bounded-degree subgraphs satisfying given connectivity properties. Given a graph G with nonnegative weights on the edges, and a degree bound Δ, their algorithm computes a spanning tree of G whose degree is at most $O(\Delta \log \frac{n}{\Delta})$ and whose weight is at most $O(\log n)$ times the weight of a minimum-weight tree with degree at most Δ. Their techniques extend to the Steiner tree and generalized Steiner tree problems with the same ratio. They also studied special cases when the edge weights satisfy the triangle inequality and presented efficient algorithms for computing subgraphs that have low weight and small bottleneck cost. More recently, Könemann and Ravi [7] have given an algorithm that finds a tree of degree $O(\Delta + \log n)$ whose cost is at most $O(1)$ times the cost of an optimal tree of degree Δ.

Given a graph G and an independent set of nodes I, the problem of finding a spanning tree that minimizes the maximum degree of any node in I is solvable in polynomial time [8]. Gavish [5] formulated the MDST problem as a mixed integer program and provided an exact solution using the method of Lagrangian multipliers. Ravi [10] presented an approximation algorithm for the problem of finding a spanning tree whose diameter plus maximal degree is a minimum.

Previous work in directed graphs. Fürer and Raghavachari [2] showed that their algorithm for computing low-degree trees further generalizes to find branchings in directed graphs. Their algorithm builds a tree in stages by taking the union of a sequence of low-degree forests (e.g., matchings), and the degree of the

resulting tree is shown to be $O(\Delta^* \log n)$. This paper improves the performance from a multiple of $\log n$ to an additive term of $\log n$.

2 Definitions and Notation

The input is an arbitrary directed graph $G = (V, E)$, and a root vertex $r \in V$. Let n be the number of vertices in G. It is assumed that r is reachable from all vertices of G. Let T^* be an optimal DMDST whose maximal degree is Δ^*.

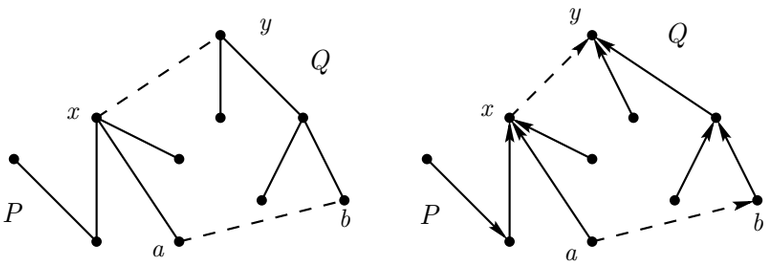
A *branching* rooted at r is a subgraph of G whose underlying undirected graph is a spanning tree such that it has a directed path from any vertex to r. In a branching, each vertex other than r has exactly one outgoing edge and r has no outgoing edges. It is easily shown that the only subgraphs with $n-1$ edges in which there is a directed path from every vertex to r is a branching rooted at r. Sometimes this is also known as an in-branching. One can also define an out-branching, in which r can reach every vertex of G through directed paths. In this paper, a branching always refers to an in-branching. However, our algorithm can be easily modified to find out-branchings with small outdegree.

Let T be a branching. For each edge (v, w) in T, we call w as the *parent* of v, denoted by $p[v]$. Since every vertex except r has a unique outgoing edge, each vertex has a unique parent, and r has none. The reflexive and transitive closure of the parent function yields the ancestor relation. In other words, v is an *ancestor* of u if there is a directed path in the branching from u to v. We call u a *descendent* of v if v is its ancestor. We say that two vertices v and w are *related* if either v is an ancestor of w, or vice versa. Otherwise, we say that the vertices are *unrelated*. For any two unrelated vertices v and w, the *least common ancestor* is the ancestor closest to v that is also an ancestor of w. We define C_v to be the set of all vertices in the subtree rooted at v, i.e., the set of all vertices including v, for which v is an ancestor.

The *degree* of a vertex in a given branching is the number of edges coming into that vertex. We may also refer to it as its *indegree*. For a branching, let S_Δ be the set of all vertices whose degree is Δ or more. The degree of a branching is the maximum degree of any of its vertices. Our goal is to find a branching of as small a degree as possible.

3 MDST Problem: Directed vs Undirected Graphs

Our algorithm is based on an algorithm proposed by Fürer and Raghavachari [3] for undirected graphs that finds a tree whose degree is $O(\Delta^* + \log n)$. Their algorithm starts with an arbitrary spanning tree of G, and iteratively decreases the degree of high-degree vertices by applying "improvement" steps. An improvement step involves replacing an edge incident to a high-degree node by another edge that keeps the tree connected. They applied improvement steps to high-degree nodes repeatedly, until no improvement was possible at these nodes. In order to extend this algorithm to directed graphs, we make several modifications.



(a) Undirected graph (b) Directed graph

Fig. 1. Directed versus undirected graphs

First, in directed graphs, we face the following problem. Suppose an edge is removed that splits the current tree into two trees, namely P and Q (see Fig. 1). Suppose we try to combine the two trees by adding the edge (x, y), where $x \in P$ and $y \in Q$. In the case of undirected graphs, any such edge would do and we get back a spanning tree of G. But in the case of directed graphs, the vertex x must be the root of the tree Q. Otherwise the procedure would not yield a branching of G. To illustrate, as shown in Fig. 1 (a), undirected trees P and Q can be merged into a single tree by adding either the edge (x, y) or the edge (a, b). But in the case of directed graphs, as shown in Fig. 1 (b), only the edge (x, y) yields a branching. If (a, b) is added, then a has two outgoing edges, and the resulting graph is not a branching. Therefore the improvement step needs to be modified.

Second, the analysis of the algorithm for undirected graphs uses the notion of "witness sets". A witness set is a small set of nodes whose removal splits the graph into a large number of connected components. The ratio of the number of components to the number of nodes in the witness set is a lower bound on Δ^*. It was shown by Fürer and Raghavachari [3] that there are witness sets that can be used to find a tree whose degree is at most $\Delta^* + 1$. We will show that the notion of a witness set must also be modified for the case of directed graphs.

3.1 Witness Sets

The minimum ratio of the cardinality of a vertex set W to the number of components that are generated when W is removed from the graph is called the *toughness* of the graph. Win [13] has shown the following interesting relationship between the toughness of a graph and the MDST problem. He showed that if the toughness of a graph is at least $\frac{1}{k-2}$, then it has a spanning tree whose degree is at most k (for $k > 2$). Vertex sets for which the above ratio is close to the toughness of the graph are called witness sets. Fürer and Raghavachari [3] used such witness sets to establish a lower bound on the degree of an optimal tree for the MDST problem. They showed that if there is a witness set of size w whose removal splits G into t components then $\Delta^* \geq \lceil \frac{w+t-1}{w} \rceil$.

This definition is not suitable for directed graphs. We prove a new lemma that can be used to establish a lower bound on Δ^* for directed graphs. Suppose we have a set of witness vertices W and a set of blocking vertices B satisfying the property that paths from different vertices of W do not intersect before being incident to a vertex in B (see Fig. 2). From this we show that there are $|W|$ paths that have distinct edges into B, thus establishing a lower bound on the degree of vertices in B in any branching.

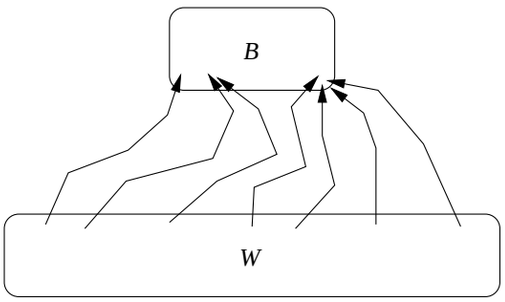


Fig. 2. Paths from W are internally disjoint

Lemma 1. *Let $G = (V, E)$ be a directed graph and $r \in V$. Suppose there are subsets of vertices $W \subset V$ and $B \subset V$ that satisfy the following properties:*

1. *Any path from a vertex $v \in W$ to r must have an incoming edge into a vertex in B,*
2. *For any two vertices $v, w \in W$, any path from v to r can intersect a path from w to r only after it passes through a vertex in B. In other words, G has no branching wherein the path from v to the least common ancestor of v and w does not contain a vertex of B.*

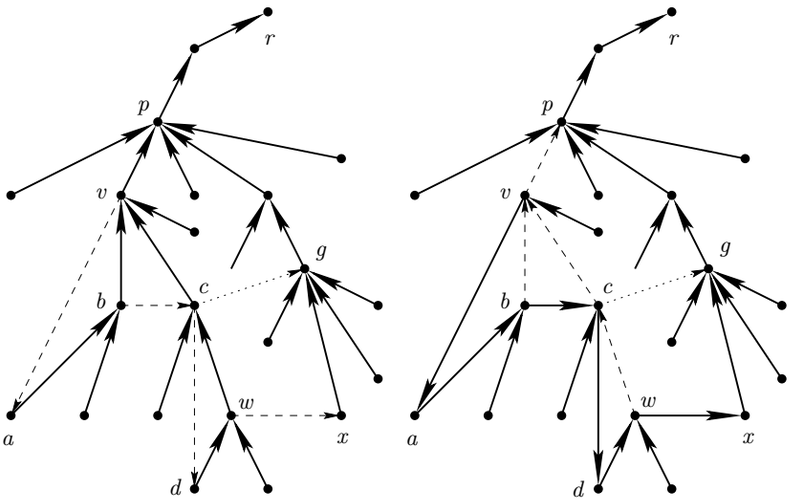
Then the degree of an DMDST rooted at r of G satisfies, $\Delta^* \geq \lceil |W|/|B| \rceil$.

Proof. Let T^* be an optimal branching rooted at r for the DMDST problem. Since it is a branching, it contains a path from any vertex to the root. By Condition 1 of the lemma, a path from a vertex $v \in W$ to r contains at least one edge into a vertex in B. Let $f_v \neq v$ be the closest ancestor of v such that $f_v \in B$. Let P_v be this path from v to f_v. By Condition 2 of the lemma, the paths $\{P_v : v \in V - \{r\}\}$ are all internally disjoint. Therefore we have identified $|W|$ paths in T^*, and each of these paths has an incoming edge to some vertex in B. Therefore the average degree of a vertex in B is at least $|W|/|B|$, implying that there is at least one vertex in T^* whose degree is $\lceil |W|/|B| \rceil$ or more.

4 The DMDST Algorithm

Our algorithm starts with an arbitrary branching T of G and reduces the degree of high-degree nodes iteratively by applying improvement steps defined below.

Consider a node v whose parent in the tree is p. We can decrease the indegree of p by 1 (which is an improvement step applied to p) if we can delete the edge (v, p) and find an alternate path for v to reach the root r. This new path from v to r initially goes through some nodes of C_v, vertices in the subtree rooted at v, reaching a node $w \in C_v$ (w may be v itself). A new edge (w, x) is added (replacing the edge from w to its parent) where x is unrelated to v. Since x is unrelated to v, it is unrelated to any vertex in C_v. Therefore the path from x to r in T is unaffected. Since v can reach x after the improvement, v can reach r. We perform an improvement step only if after the improvement, vertices whose degrees increased have a smaller degree than p.



(a) Branching before improvement (b) Branching after improvement

Fig. 3. Example of an improvement applied to vertex p

Fig. 3 illustrates an example of an improvement step. In this example, the tree edges are shown in thick lines and other edges of g are shown in dashed lines. The indegree of p is 5. If v can find an alternate path to r so that the edge (v, p) may be deleted from T, the degree of p can be decreased to 4. The edge (c, g) is deleted because the indegree of g is already 4, and if we choose to add this edge, its indegree becomes 5. Decreasing the degree of p to 4 by increasing the degree of g to 5 (old degree of p) does not make progress. The edge (v, p) is also deleted and the algorithm tries to find a path from v to r. Such a path exists — $(v \to a \to b \to c \to d \to w \to x \to \ldots \to r)$ and the algorithm uses this path to modify the branching; the new branching is shown in Fig. 3 (b). The indegree of p has thus been successfully reduced to 4.

We will now describe how to test if such an improvement exists. Let the degree of p be Δ. We first ensure that the degree of vertices whose degree is $\Delta - 1$ or

greater does not increase. Delete all nontree edges of G that are incident into nodes of degree $\Delta - 1$ or greater, i.e., $S_{\Delta-1}$. In the remaining graph, delete the edge (v, p) and test if there is a path from v to r. If such a path exists, we can select a shortest such path and use it to make an improvement to p as follows. Let x be the vertex closest to v in the path such that $x \notin C_v$. For each edge (y, z) in the path from v to x, we replace the edge $(y, p[y])$ by the edge (y, z). It can be verified that the above operation results in another branching since the number of edges is still $n - 1$ and all vertices can still reach r.

Procedure IMPROVEMENT(T, v, p)

1. Delete (v, p) from G.
2. Let Δ be the degree of p. For each vertex $u \in V$ whose indegree in T is greater than $\Delta - 1$, delete from G edges going into u that are not in T.
3. Run Breadth-first search from v, and test if the root r is reachable from v.
4. If there is no path from v to r, return False after restoring all edges of G.
5. Otherwise, BFS finds a path P from v to r. Let w be the first vertex on the path with the property that $(w, x) \in P$ and $w \in C_v$ and $x \notin C_v$.
6. For each edge (a, b) in the subpath of P from v to x, replace the edge from $(a, p[a])$ in T by (a, b).
7. Restore all edges of G and return True.

We now consider the DMDST algorithm. The algorithm tries to reduce the degree of high-degree vertices by finding suitable improvements. The target vertices are those whose degrees are within $O(\log n)$ from the maximal degree of the current branching. When no improvements are possible to these nodes, the algorithm terminates.

Algorithm DMDST(G, r)

1. Find a branching T of G rooted at r. Let its degree be k. Fix some constant $c > 1$.
2. For each edge $(v, p) \in T$, run IMPROVEMENT(T, v, p) if the degree of p in T is more than $k - \lceil \log_c n \rceil$. If the degree of T has changed, reset k to be its new degree.
3. Repeat the above step until IMPROVEMENT(T, v, p) returns false for every edge $(v, p) \in T$ for which it is called.
4. Return T.

5 Analysis of the Algorithm

The analysis of the running time of the algorithm uses potential functions that were introduced by Fürer and Raghavachari [3], and adapted by Ravi, Raghavachari and Klein [12] and Könemann and Ravi [7]. In fact our analysis of the running time is almost the same as in [12]. The potential of a vertex of degree Δ is defined to be n^{Δ}, and therefore the total potential of all the vertices

is at most n^{k+1}, where k is the current degree of T. An improvement is applied to a vertex of degree $\Delta > k - \lceil \log_c n \rceil$. Each improvement step that targets a vertex of degree Δ reduces the total potential by at least $n^{\Delta-2}$, since the degree of a node of degree Δ is reduced by 1 and the degree of all the other nodes may increase to $\Delta - 1$. Since $\Delta > k - \lceil \log_c n \rceil$, this reduction in potential is at least a fraction $n^{-\log_c n - 3}$ of the current potential of the branching. It follows that the number of improvement steps is at most $n^{\log_c n + 3}$. Each improvement step can be implemented in $O(n^3)$ time, thus giving a total running time of $O(n^{\log_c n + 6})$ for the algorithm.

The following lemma relates the running time of the algorithm and the number of improvement steps, I. This expression for the running time is a more meaningful measure, since our experiments show that I grows only linearly with n, making the observed running time $O(n^4)$.

Lemma 2. *The running time of Algorithm DMDST is $O(n^3 I)$, where I is the number of improvement steps.*

The analysis of the degree bound of the tree output by the algorithm is more interesting. For this analysis, the notion of witness sets that was used by the algorithm for undirected graphs has to be strengthened. In directed graphs, there may be edges in the "wrong" direction that don't help in constructing a branching, but they may stop the graph from falling apart when a few critical vertices are removed.

We now show how to find a witness set W and its blocking set B for the branching T output by our algorithm. In fact we will identify one pair of sets W and B for each Δ in the range k to $k - \lceil \log_c n \rceil$.

Lemma 3. *Let T be a branching whose degree is Δ or more. Let S_Δ be the set of vertices whose degree is Δ or more. There are at least $(\Delta - 1)|S_\Delta| + 1$ unrelated vertices such that the parent of each of these vertices is in S_Δ.*

Proof. The proof is by induction on the cardinality of S_Δ. If $|S_\Delta| = 1$, then the single vertex in that set has at least Δ children, and the children of this vertex satisfy the lemma. If $|S_\Delta| > 1$, remove a node $v \in S_\Delta$ and all its descendents from T such that v has no descendents in S_Δ (except itself). Now the resulting branching has $|S_\Delta| - 1$ nodes of degree Δ or more, and by the induction hypothesis, has at least $(\Delta - 1)(|S_\Delta| - 1) + 1$ unrelated nodes that are children of S_Δ. Since all these nodes are unrelated to each other, at most one of these nodes is an ancestor of v. Therefore there are $(\Delta - 1)(|S_\Delta| - 1)$ nodes left that are not ancestors of v. Now we add the children of v to this set, the set increases by at least Δ, and the number of nodes that we get is $(\Delta - 1)(|S_\Delta| - 1) + \Delta = (\Delta - 1)|S_\Delta| + 1$.

Lemma 4. *Let T be the branching output by our algorithm. Let its degree be k. Then for any $k - \lceil \log_c n \rceil < \Delta \leq k$,*

$$\Delta^* \geq \frac{(\Delta - 1)|S_\Delta| + 1}{|S_{\Delta - 1}|}.$$

Proof. Let W be the set of vertices as in Lemma 3 that are children of nodes in S_Δ, but have no descendents in S_Δ. We know that $|W| \geq (\Delta - 1)|S_\Delta| + 1$. Let B be $S_{\Delta-1}$, the set of all vertices whose degree is at least $\Delta - 1$. For each vertex $v \in W$, the algorithm tries to find an improvement that decreases the degree of $p = p[v]$. Since it failed (the condition under which the algorithm stops), any path from v to r that doesn't use (v, p) must go through a vertex x in $S_{\Delta-1}$. By construction, the internal vertices of the path from v to x is entirely contained in C_v, the descendents of v in T. Since all vertices of W are unrelated to each other, these subtrees are disjoint. Therefore, the sets W and B that we have defined satisfy the conditions given in the statement of Lemma 1. Therefore,

$$\Delta^* \geq \lceil |W|/|B| \rceil \geq \frac{(\Delta - 1)|S_\Delta| + 1}{|S_{\Delta-1}|}.$$

Theorem 1. *The degree of the branching returned by our algorithm is at most* $c\Delta^* + \log_c n$*, where* $c > 1$ *is the constant in Step 1 of the DMDST algorithm.*

Proof. Lemma 4 establishes a set of lower bounds on Δ^* for $\lceil \log_c n \rceil$ different values of Δ. At least for one of these values of Δ, $|S_{\Delta-1}| \leq c|S_\Delta|$. Using this value of Δ, we get $k \leq c\Delta^* + log_c n$.

6 Experimental Results

The algorithms were implemented in C, using Knuth's Stanford GraphBase toolkit [6] and tested on large numbers of randomly generated graphs. These input random graphs actually followed two different patterns, as described below individually. The running time clearly depends upon the initial tree, and this was indeed observed in the experimental study. We tried to generate the initial tree using both depth-first search (DFS) and breadth-first search (BFS). BFS tends to generate high degree nodes and therefore the number of improvement steps tends to be much higher than if the initial tree was generated by DFS. In dense graphs, DFS generally finds low-degree trees by itself. While we found that our algorithm further reduces the degree of the initial DFS tree significantly, we present experimental results with an initial BFS tree, because we wished to capture the worst-case performance of the algorithm.

6.1 Uniformly Distributed Random Graphs

For this class of randomly generated input graphs, the probability of existence of an edge between two vertices is set to be a constant. The number of vertices were varied from 100 to 9000. A small number of runs were on 20,000-node graphs. We also varied the density of the graph. This class of graphs tends to be Hamiltonian and a good algorithm should be able to find a low-degree branching. We observed that our algorithm also has no difficulty in achieving this.

The results for this class of graphs is presented in Fig. 4, which shows the number of improvement steps as a function of n. The x-axis shows n and the

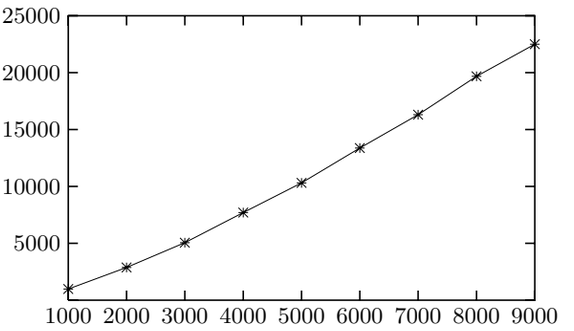


Fig. 4. Number of improvement steps versus n

y-axis shows the average number of improvement steps in random graphs with uniform edge probability. For each value of n, the algorithm was run on several random instances and the average number of improvement steps over these instances was used to generate this plot. Note that the number of improvement steps is almost linear in n.

In all of our test cases, we found that the algorithm always found a branching of degree two or less, irrespective of how large n was or how bad the initial degree of the tree was. For problems of small size (n less than 100) the algorithm would often return a branching of degree one, i.e., a Hamiltonian path.

Fig. 5 shows the degree of the intermediate trees as the algorithm progresses on a 20,000-node random graph with uniform edge probability. Observe that the algorithm makes rapid progress initially since there are very few vertices of high degree. Once the degree becomes small, a larger number of improvements are needed to decrease the degree of the tree. Note that if one terminated the algorithm earlier (say, to meet a fixed deadline) then the current tree can be used without a big sacrifice on the quality. The shape of the curve shows that we get about 50% of the progress in about 10% of the time.

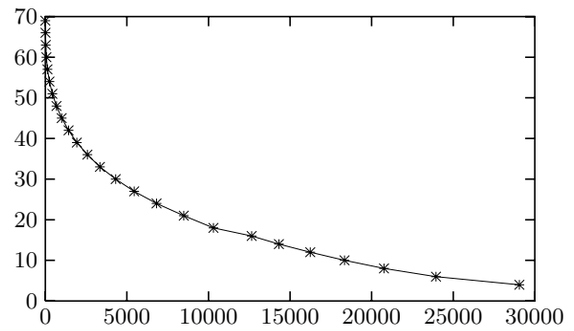


Fig. 5. Degree of tree versus number of improvements on a 20,000 node graph

6.2 Graphs with a Hidden Hamiltonian Path

The second class of graphs we tested were deliberately constructed to provide "bad" inputs to the algorithm to really test the algorithm's power and efficacy. The graphs in this class were generated by first obtaining a random bipartite graph with n_1 vertices on one side and n_2 vertices on the other side (say $n_1 < n_2$). The ratio of n_2/n_1 was varied (while holding $n = n_1 + n_2$ constant). We finally added to each of these bipartite graphs a random Hamiltonian path. Without the "hidden" path that we added at the end, the degree of any branching in the bipartite graph is at least n_2/n_1, since vertices in the tree must alternate between the two sides. We wanted to see if the algorithm finds this hidden path.

The algorithm performed very well even in these bad input instances, always returning no more than a degree 2 tree in the end. As shown in Fig. 6 the number of improvement steps varies significantly with the ratio n_2/n_1, a measure of the badness of the input graph. As expected, the algorithm takes longer as the ratio gets closer to 1. We observed that in all cases, the number of improvements is a small multiple of n.

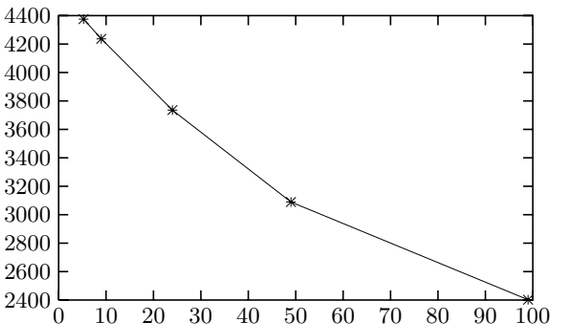


Fig. 6. The x-axis shows the ratio of n_2/n_1 and the y-axis shows the average number of improvement steps for bipartite graphs with a total of 1000 vertices. The input graphs for this plot were randomly generated bipartite graphs that were augmented with a hidden Hamiltonian path.

7 Conclusions

We have presented an approximation algorithm for the directed minimum-degree spanning tree problem. We introduced a new notion of witness sets that works in directed graphs. Though we couldn't prove a polynomial running time for our algorithm, it is likely to be fast in practice as shown by the experimental evidence. There are several open questions that follow from this work. Is it possible to implement our algorithm to run in polynomial time? Currently, the Steiner version of the DMDST problem does not have an approximation algorithm even with an $O(\log n)$ approximation ratio.

We describe an example that shows why our current approach fails in the Steiner version of the DMDST problem. In this example, there is a high-degree node p of degree k. Its children are $c_1, \ldots, c_k$. Each of these k children have an edge into vertex s, which is not in the current Steiner tree, and s has an edge into p. It can be verified that there is no improvement possible for vertex p. But, the degree of p can be reduced to $\lfloor \frac{k}{2} \rfloor + 1$ by connecting $\lfloor \frac{k}{2} \rfloor$ of p's children through s. In fact if the graph has a number of other extra nodes similar to s, the degree of p can be reduced even to 2. This example shows that our algorithm does not guarantee any performance bound on the degree of the tree for the Steiner case. The reason that we were unable to apply Lemma 1 is that, the paths from c_1 through c_k (the nodes in W) to r intersect each other at s, before reaching p (which forms the set B), thus violating Condition 2 of the lemma.

References

1. A. Agrawal, P. Klein, and R. Ravi, How tough is the minimum-degree Steiner tree? A new approximate min-max equality, TR CS-91-49, Brown University, 1991.
2. M. Fürer and B. Raghavachari, An NC approximation algorithm for the minimum degree spanning tree problem, In *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing*, pages 274–281, 1990.
3. M. Fürer and B. Raghavachari, Approximating the minimum degree spanning tree to within one from the optimal degree, In *Proc. of 3rd ACM-SIAM Symp. on Disc. Algorithms (SODA)*, pages 317–324, 1992.
4. M. Fürer and B. Raghavachari, Approximating the minimum-degree Steiner tree to within one of optimal, *J. Algorithms*, 17:409–423, 1994.
5. B. Gavish, Topological design of centralized computer networks – formulations and algorithms, *Networks*, 12:355–377, 1982.
6. D. E. Knuth, The Stanford GraphBase: a platform for combinatorial computing, Addison Wesley, 1993.
7. J. Könemann and R. Ravi, A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees, In *Proc. of 32nd annual ACM Symposium on Theory of Computing (STOC)*, pages 537-546, 2000.
8. E. L. Lawler, Combinatorial optimization: networks and matroids, Holt, Rinehart and Winston, New York, 1976.
9. B. Raghavachari, Algorithms for finding low degree structures, In "Approximation algorithms," D. Hochbaum (ed.), PWS Publishers Inc., pages 266-295, 1996.
10. R. Ravi, Rapid rumor ramification: approximating the minimum broadcast time, In *Proc. of 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 202–213, 1994.
11. R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, Many birds with one stone: multi-objective approximation algorithms, In *Proc. of 25th Annual ACM Symp. on the Theory of Computing (STOC)*, pages 438–447, 1993.
12. R. Ravi, B. Raghavachari, and P. Klein, Approximation through local optimality: designing networks with small degree, In *Proc. of 12th Conf. on Foundations of Software Tech. and Theoret. Comp. Sci. (FSTTCS)*, pages 279–290. Lect. Notes in Comp. Sci. 652, 1992.
13. S. Win, On a connection between the existence of k-trees and the toughness of a graph, *Graphs and Combinatorics*, 5:201-205, 1989.

I/O-Efficient Batched Range Counting and Its Applications to Proximity Problems*

Tamás Lukovszki[1], Anil Maheshwari[2], and Norbert Zeh[2]

[1] Heinz-Nixdorf-Institut, Universität Paderborn, Germany
tamas@hni.uni-paderborn.de
[2] School of Computer Science, Carleton University, Ottawa, Canada
{maheshwa,nzeh}@scs.carleton.ca

Abstract. We present an algorithm to answer a set Q of range counting queries over a point set P in d dimensions. The algorithm takes $O\left(\text{sort}(|P|+|Q|) + \frac{(|P|+|Q|)\alpha(|P|)}{DB}\log_{M/B}^{d-1}\frac{|P|+|Q|}{B}\right)$[1] I/Os and uses linear space. For an important special case, the $\alpha(|P|)$ term in the I/O-complexity of the algorithm can be eliminated. We apply this algorithm to constructing t-spanners for point sets in $\mathbb{R}^d$ and polygonal obstacles in the plane, and finding the K closest pairs of a point set in $\mathbb{R}^d$.

1 Introduction

Motivation. Range searching and range counting problems have applications to spatial databases, geographic information systems, statistical analysis, and problems in computational geometry [1]. Conical Voronoi diagrams and θ-graphs appeared first in the early 80's as the base structure of the so called *region approach* for solving nearest neighbor problems and constructing a minimum spanning tree for a given set S of N points in d-dimensional Euclidean space [23]. These structures have numerous further applications and a rich history in various fields of computer science, e.g., in motion planning [11], construction of spanner graphs [16,18,19], problems on communication networks [15], for approximating the Euclidean minimum spanning tree [23] of a given point set, and for real-time walkthrough in virtual scenes [12]. In most of these domains, realistic data sets are too large to fit into the internal memory of state-of-the-art computers. Therefore special techniques are required to reduce the overhead involved in accessing secondary storage (i.e., disks). Existing internal memory techniques cannot be adapted, as random access into secondary memory is far too expensive.

Range counting. Range counting is a special kind of range searching problem. Given a point set P and a query range $q = [x_1, x_1'] \times [x_2, x_2'] \times \cdots \times [x_d, x_d']$, the standard range searching problem is to report all points in $P \cap q$. It is easy

* Research supported by NSERC, NCE GEOIDE, and DFG-SFB376.

[1] sort(N) denotes the I/O-complexity of sorting N data items in external memory. See *Model of Computation*.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 244–255, 2001.
© Springer-Verlag Berlin Heidelberg 2001

to answer a single query of this type in optimal $\Theta(\text{scan}(N))$ I/Os. Typically, however, we are either presented with a batch of queries (the *batched* scenario), or we want to build a data structure that can answer queries in $o(\text{scan}(N))$ I/Os per query, depending on the size of the output (the *online* scenario). I/O-efficient solutions for the online range searching problem have been presented in [3,7,20]. When solving the batched range searching problem, our goal is to minimize the total number of I/Os spent on answering all queries. An I/O-efficient solution to this problem has been presented in [2].

While range searching asks to report all points in $P \cap q$, range counting asks to report a value $\bigotimes_{p \in P \cap q} \lambda(p)$, where $\otimes$ is a commutative and associative operator and $\lambda(p)$ is a label assigned to point p. Important special cases include counting the elements in $P \cap q$ (the labels are 1, and $\otimes$ is standard addition) or finding the minimum point in $P \cap q$ w.r.t. some arbitrary weighting scheme.

θ-Graphs. Given a set $\{p_0, \ldots, p_k\}$ of points in $\mathbb{R}^d$ such that the vectors $(p_i - p_0)$, $0 \leq i \leq d$ are linearly independent, we define the *simplicial cone* spanned by points $p_0, \ldots, p_k$ as the set $\{p_0 + \sum_{i=1}^{d} \lambda_i (p_i - p_0) : \lambda_i \geq 0\}$. We call p_0 its *apex*. A *θ-frame* is a set C of simplicial cones such that each cone has its apex at the origin, $\bigcup_{c \in C} c = \mathbb{R}^d$, and each cone $c_i \in C$ contains a ray l_i emanating from the origin such that for any other ray l in the cone emanating from the origin, $\angle l_i l \leq \theta/2$. Denote l_i as the *cone axis* of c_i. In [19], it is shown how to construct a θ-frame of size $(d/\theta)^{O(d)}$.

Given a θ-frame C and a point p, let $c_0(p), \ldots, c_{k-1}(p)$ and $l_0(p), \ldots, l_{k-1}(p)$ be translated copies of cones $c_0, \ldots, c_{k-1} \in C$ and rays $l_0, \ldots, l_{k-1}$ such that the apexes of cones $c_i(p)$ and the endpoints of rays $l_i(p)$ coincide with point p. For $p, q \in P$ and $0 \leq i < k$, the distance $dist_{c_i}(p, q)$ between p and q w.r.t. cone c_i is defined as the Euclidean distance between p and the projection of q onto the translated ray $l_i(p)$ if q is contained in $c_i(p)$, and infinity otherwise. The Euclidean distance between two points p and q is denoted by $dist_2(p, q)$. For each point $p \in P$ and $0 \leq i < k$, let $P_{c_i(p)} = P \cap c_i(p)$.

The *K-th order θ-graph* $G_{\theta,K}(P)$ is defined as follows: The points of P are the vertices of $G_{\theta,K}(P)$. For every point $p \in P$ and every cone $c_i \in C$, we add directed edges from p to the $K^* = \min(K, |P_{c_i(p)}|)$ vertices in $P_{c_i(p)}$ that are closest to p w.r.t. the distance function $dist_{c_i}$.

A *t-spanner*, $t > 1$, for a point set P is a straight line graph with vertex set P such that for each pair $p, q \in P$ of vertices, the shortest path from p to q in G is at most t times longer than the Euclidean distance from p to q; the *length* of a path is the sum of the Euclidean lengths of its edges. We call such a shortest path a *t-spanner path* from p to q; t is called the *stretch factor* of G. In [19], it is shown that in a fixed dimension $d \geq 2$ and for $0 < \theta < \frac{\pi}{3}$, the θ-graph $G_\theta(P)$ is a $\left(\frac{1}{1-2\sin(\theta/2)}\right)$*-spanner* for P, and that it can be constructed in $O(N \log^{d-1} N)$ time using $O(N \log^{d-2} N)$ space.

A spanner graph G is *K-fault tolerant* if after removing at most K vertices or edges from G, the remaining graph still contains a path between each pair p, q of vertices which is at most t times longer than the shortest path between p

and q in the complete Euclidean graph after removing the same set of vertices or edges [17]. In [18], it is shown that the K-th order θ-graph is K-fault tolerant.

The θ-graph $G_\theta(O)$ for a set O of simple polygonal obstacles in the plane is defined as follows: The vertex set of $G_\theta(O)$ is the set of obstacle vertices in O. Each vertex v is connected to the nearest *visible* vertex in each cone $c(v)$ w.r.t. $dist_c$. This graph has been introduced in [11] to solve the approximate shortest path problem in two and three dimensions: Given two points s and t and a constant $\epsilon > 0$, find an obstacle avoiding path which is at most $(1 + \epsilon)$ times longer than the shortest obstacle avoiding path between s and t.

Conical Voronoi diagrams. The *conical Voronoi diagram* $CVD_c(P)$ of a set P of points is closely related to the θ-graph. For a cone c, the *conical Voronoi region* V_p of a point $p \in P$ is the set of points in the plane having p as the closest point in P w.r.t. $dist_c$. $CVD_c(P)$ is the planar subdivision defined by the Voronoi regions V_p, $p \in P$. Similarly, the conical Voronoi diagram $CVD_c(O)$ of a set O of simple polygonal obstacles is the planar subdivision defined by the obstacles and the Voronoi regions V_p of the obstacle vertices, where V_p contains the points $x \in \mathbb{R}^2$ having obstacle vertex p as the closest *visible* vertex w.r.t. $dist_c$.

Model of computation and related results. Our algorithms are designed and analyzed in the Parallel Disk Model (PDM) [22]: An external memory consisting of D disks is attached to a machine with an internal memory of size M. Each of the D disks is divided into blocks of B consecutive data items. Up to D blocks, at most one per disk, can be transferred between internal and external memory in a single I/O-operation. The complexity of an algorithm is the number of I/O operations it performs. Many external memory (EM) algorithms and techniques for fundamental problems in computational geometry, graph theory, GIS, matrix computations, etc. have been developed in this model. Due to the lack of space we refer the reader to the survey of [21] and mention only the most relevant work here. It has been shown that sorting an array of size N takes $\text{sort}(N) = \Theta\left(\frac{N}{DB}\log_{M/B}\frac{N}{B}\right)$ I/Os [21,22]; scanning an array of size N takes $\text{scan}(N) = \Theta\left(\frac{N}{DB}\right)$ I/Os. EM algorithms for computing pairwise intersections of orthogonal line segments, answering range queries in the plane, finding all nearest neighbors for a set of N points in the plane, dominance problems, and other geometric problems in the plane are discussed in [2,3,7,13,20]. General line segment intersection problems have been studied in [6]. For lower bounds on computational geometry problems in EM see [5]. See [4] for buffer trees, priority queues, and their applications.

Overview. In Sect. 2, we discuss our solution to the batched range counting problem. In Sect. 3, we use the solution for a special case of this problem to compute K-th order θ-graphs for point sets in d dimensions. We also discuss how to report a spanner path I/O-efficiently. In Sect. 4, we apply $O(\sqrt{K})$-th order θ-graphs to solve the K-closest pairs problem in d dimensions. Finally, in

Sect. 5, we show how to compute the θ-graph and the conical Voronoi diagram of a given set of disjoint simple polygonal obstacles in the plane I/O-efficiently.

In order to solve the batched range counting problem, one could use the range searching algorithm of [2] to report the points in each query range, and then count the points reported for each query. Using this strategy, the complexity of answering a query depends on the number of points in the query range, which can be as large as N. Our solution is independent of the number of points in each query range.

In [14], an asymptotically more efficient construction for spanners of point sets in d dimensions is presented. However, as this construction is based on a well-separated pair decomposition [10] of the point set, the constants hidden in the big-Oh notation are extremely large, the construction works only for point sets, and the approach cannot be used to construct K-fault tolerant spanners. For moderate dimensions, we expect our algorithm to compare favorably with the one of [14]. Similarly, the solution to the K-closest pair problem presented in [14] is asymptotically more efficent than ours; but for moderate dimensions we expect our algorithm to be more efficient.

Our construction of θ-graphs for sets of polygons is based on the construction of [11]. However, the algorithm of [11] is not I/O-efficient and cannot easily be made I/O-efficient. We prove a number of interesting properties of conical Voronoi diagrams to obtain an I/O-efficient solution to this problem.

2 Batched Higher-Dimensional Range Counting

In this section, we present I/O-efficient algorithms for the batched d-dimensional range counting problem. First we consider the important special case where w.l.o.g. $x'_1 = \infty$, for all query ranges $[x_1, x'_1] \times \cdots \times [x_d, x'_d]$. Our solution for this case is used in Sections 3 and 5. For the general case, we present a more complicated algorithm, which is by a factor of $O(\alpha(|P|))$ slower.

2.1 Colorable Problems

To solve the batched range counting problem, we apply the framework of colorable search problems defined in [2], although we extend it slightly. This framework can be used to derive an algorithm solving a search problem in $\mathbb{R}^d$, $d > 1$, from an algorithm solving the same search problem in $\mathbb{R}^1$.

Let $\mathcal{P}$ be a batched search problem answering a set Q of queries over a point set P in $\mathbb{R}^d$. Given a set C of colors, we define a coloring $\mathcal{C}$ assigning a color $c_q \in C$ to every query $q \in Q$ and a set $C_p \subseteq C$ of colors to every point $p \in P$. Every color $c \in C$ defines a point set $P_c = \{p \in P : c \in C_p\}$. Let $\mathcal{P}_\mathcal{C}$ be the problem of answering queries $q \in Q$ with respect to point sets P_{c_q}.

We call $\mathcal{P}$ $(\mathcal{I}_p, \mathcal{I}_r, S)$-*$m^c$-colorable* in dimension d, for some constant $0 < c \leq 1$, if for every coloring $\mathcal{C}$ with $|C| = O\big(\sqrt{m^c}\big)$ and such that there are $O(m^c)$ different color sets assigned to the points in P, there exists an algorithm $\mathcal{A}$ that solves problem $\mathcal{P}_\mathcal{C}$ and can be divided into phases $\mathcal{A}_p^{(1)}, \mathcal{A}_r^{(1)}, \mathcal{A}_p^{(2)}, \mathcal{A}_r^{(2)}, \ldots,$

$\mathcal{A}_p^{(k-1)}, \mathcal{A}_r^{(k-1)}, \mathcal{A}_p^{(k)}$ so that phases $\mathcal{A}_p^{(1)}, \ldots, \mathcal{A}_p^{(k)}$ take $\mathcal{I}_p$ I/Os in total, the total I/O-complexity of phases $\mathcal{A}_r^{(1)}, \ldots, \mathcal{A}_r^{(k-1)}$ is $\mathcal{I}_r$, $\mathcal{A}$ uses S space, and phases $\mathcal{A}_p^{(1)}, \ldots, \mathcal{A}_p^{(k)}$ are independent of d.

The idea behind this definition is that we can use algorithm $\mathcal{A}$ to derive an algorithm $\mathcal{B}$ solving $\mathcal{P}$ in $\mathbb{R}^{d+1}$, only by replacing phases $\mathcal{A}_r^{(1)}, \ldots, \mathcal{A}_r^{(k-1)}$ by phases $\mathcal{B}_r^{(1)}, \ldots, \mathcal{B}_r^{(k-1)}$, in order to deal with the extra dimension. That is algorithm $\mathcal{B}$ consists of phases $\mathcal{A}_p^{(1)}, \mathcal{B}_r^{(1)}, \mathcal{A}_p^{(2)}, \mathcal{B}_r^{(2)}, \ldots, \mathcal{A}_p^{(k-1)}, \mathcal{B}_r^{(k-1)}, \mathcal{A}_p^{(k)}$.

We solve the search problem to be addressed by phase $\mathcal{B}_r^{(i)}$ using a buffer tree [4] of degree $\sqrt{m^c}$ and algorithm $\mathcal{A}_r^{(i)}$. The buffer tree is built over the coordinates of the points in P in the d-th dimension. Queries are filtered from the root of the tree to the leaves. At every level, each query q is answered w.r.t. the maximal multislab spanned by q. A point is colored with the colors of the multislabs that it is contained in. Hence, we have to solve a colored version of the d-dimensional problem at every level, which we do using $\mathcal{A}_r^{(i)}$. Adapting the proof of [2] to this more general framework, we obtain the following result.

Theorem 1. *If a batched search problem $\mathcal{P}$ is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable in $\mathbb{R}^d$, then its $(d+1)$-dimensional version is $\left(\mathcal{I}_p, \mathcal{I}_r \cdot \log_{M/B} \frac{|P|}{B}, S\right)$-$\sqrt{m^c}$-colorable.*

Corollary 1. *If a batched search problem $\mathcal{P}$ is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable in $\mathbb{R}^1$, then its d-dimensional version can be solved in $O\left(\mathcal{I}_p + \mathcal{I}_r \cdot \log_{M/B}^{d-1} \frac{|P|}{B}\right)$ I/Os using $O(S)$ blocks of external memory.*

We now apply this framework to solve the batched range counting problem.

2.2 Partially Unbounded Queries

The following algorithm shows that if w.l.o.g. $x_1' = +\infty$, for all queries $q \in Q$, the batched range counting problem is $\left(\text{sort}(|P|+|Q|), \text{scan}(|P|+|Q|), \frac{|P|+|Q|}{B}\right)$-$m$-colorable for $d = 1$: Sort all queries by left endpoints and all points by their x_1-coordinates (Phase $\mathcal{A}_p^{(1)}$). Scan P and Q in lock-step fashion, simulating a line sweep from $+\infty$ to $-\infty$. During the sweep, maintain a value $\Pi = \bigotimes\{\lambda(p) : p \in P \text{ and } x_1(p) \geq x_1(\ell)\}$, where $x_1(\ell)$ is the current position of the sweep line. When the sweep line passes a point p, update $\Pi_{\text{new}} \leftarrow \lambda(p) \otimes \Pi_{\text{old}}$. When the sweep line passes the left endpoint of a query q, report Π as the answer to query q (Phase $\mathcal{A}_r^{(1)}$). In order to make this solution m-colorable, maintain m separate products Π_{C_i}, one per color class C_i in the coloring. In order to report the answer to query q, compute $\Pi = \bigotimes\{\Pi_{C_i} : c_q \in C_i\}$ when the sweep passes the left endpoint of q. Using Cor. 1, we obtain the following result.

Theorem 2. *It takes $O\left(\text{sort}(|P|+|Q|) + \frac{|P|+|Q|}{DB} \log_{M/B}^{d-1} \frac{|P|}{B}\right)$ I/Os and linear space to answer a set Q of range counting queries over a point set P in $\mathbb{R}^d$, provided that w.l.o.g. $x_1' = +\infty$, for all queries $[x_1, x_1'] \times \cdots \times [x_d, x_d'] \in Q$.*

2.3 Bounded Queries

We turn to the general case now, where every query $q \in Q$ is a d-dimensional box. Again, we present a solution for the case $d = 1$ and generalize it to higher dimensions by applying Cor. 1. We define a sequence of functions $\phi_i(x)$ as follows: $\phi_0(x) = \lceil \frac{x}{2} \rceil$ and $\phi_i(x) = \min\{j \geq 0 : \phi_{i-1}^{(j)}(x) \leq B\}$, for $i > 0$, where $f^{(i)}(x)$ is defined as $f^{(0)}(x) = x$ and $f^{(i)}(x) = f(f^{(i-1)}(x))$, for $i > 0$. Thus, $\phi_1(x) = \log_2 x$, $\phi_2(x) = \log^* x$, and so on. It is an exercise to show that $\phi_{\alpha(N)}(N) = O(\alpha(N))$, where $\alpha(N)$ is the inverse of Ackermann's function. We develop a series of algorithms proving the following lemma.

Lemma 1. *For every $k > 0$, the one-dimensional batched range counting problem is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable, where $\mathcal{I}_p \leq tk(\text{sort}(|Q|)+\text{scan}(|P|))+t\cdot\text{sort}(|P|)$, $\mathcal{I}_r \leq tk(\text{scan}(|Q|) + \text{scan}(|P|)\phi_k(|P|))$, for some constant $t > 0$, and $S = O\left(\frac{|P|+|Q|}{B}\right)$.*

Proof sketch. Consider the case $k = 1$. The first phase $\mathcal{A}_p^{(1)}$ preprocesses P and Q as follows: Sort the points in P by their x_1-coordinates. Let T be a balanced binary tree over the points in P. (We do not construct T, but use it only as a conceptual tool.) With every node $v \in T$, associate a value x_v separating the points in the left subtree from the points in the right subtree. Associate every query $q \in Q$ with the highest node $v_q \in T$ such that q spans x_{v_q}. Split query q into two parts q_l and q_r to the left and right of x_{v_q}. This produces two sets Q_l and Q_r of left and right subqueries. Sort the queries in Q_l according to their corresponding nodes v_q, sorted top-down in T and left to right at every level; queries associated with the same node v are sorted by their left endpoints. The queries in Q_r are sorted analogously; but sort the queries associated with the same node by their right endpoints. This phase takes $O(\text{sort}(|Q|) + \text{scan}(|P|))$ I/Os and linear space.

Phase $\mathcal{A}_r^{(1)}$ answers left subqueries at each level h of T using a modification of the approach of Sect. 2.2. In particular, the running product Π is reset whenever the sweep passes a value x_v associated with a node v at level h. This takes $O(\text{scan}(|Q_h| + |P|))$ I/Os, where Q_h is the set of queries associated with nodes at level h. Right subqueries are answered using a similar procedure. As there are $\log_2 |P| = \phi_1(|P|)$ levels in T, and $\sum_{i=0}^{\log_2 |P|} |Q_h| = |Q|$, it takes $O(\text{scan}(|Q|) + \text{scan}(|P|)\phi_1(|P|))$ I/Os to answer all left and right subqueries. Once this is done, Phase $\mathcal{A}_p^{(2)}$ combines the answers to the left and right subqueries of each query in $O(\text{sort}(|Q|))$ I/Os. Choosing t appropriately, we obtain the claim for $k = 1$.

For $k > 1$, we define the tree T as follows: First associate the whole set P with the root r of T. Then split P into $|P|/\phi_{k-1}(|P|)$ subsets of size $\phi_{k-1}(|P|)$, and create one child of r for each subset. Apply this strategy recursively to the children of r. Consider a node v with children $w_0, \ldots, w_s$. Let $x_1, \ldots, x_s$ be values such that x_i separates the points associated with w_{i-1} from the points associated with w_i. Again, associate a query $q \in Q$ with the highest node $v_q \in T$ such that q spans some value x_i. Let x_l and x_r be the leftmost and rightmost

such values spanned by q, respectively. Split q into three subqueries q_l, q_m, and q_r to the left of x_l, between x_l and x_r, and to the right of x_r, respectively. Note that q_m does not exist if $x_l = x_r$. Now sort left and right subqueries by level and within each level as for the case $k = 1$. This modified Phase $\mathcal{A}_p^{(1)}$ still takes at most $(t/2)(\text{sort}(|Q|) + \text{scan}(|P|))$ I/Os.

Phase $\mathcal{A}_r^{(1)}$ now answers left and right subqueries as for $k = 1$. As the height of T is now $\phi_k(|P|)$, this takes at most $t(\text{scan}(|Q|) + \text{scan}(|P|)\phi_k(|P|))$ I/Os. Phase $\mathcal{A}_p^{(2)}$ combines the query results of q_l and q_r, and stores the result with q_m, in order to combine it with the answer to query q_m later. This takes at most $(t/2)\text{sort}(|Q|)$ I/Os. The remainder of the algorithm answers the middle subqueries of all queries in Q. Note that at every level of T, middle subqueries now stretch between values x_i. Call the interval bounded by two consecutive such values at the same level a *slab*. In order to answer middle subqueries at a particular level h, we compute a new point set P_h containing one point p_σ per slab σ. The label associated with p_σ is the product of the labels of all points in $P \cap \sigma$. We can now answer middle subqueries with respect to P_h without altering the solution. Due to the reduced size of P_h this takes $\mathcal{I}'_p \leq t(k-1)(\text{sort}(|Q_h|) + \text{scan}(|P_h|))$ and $\mathcal{I}'_r \leq t(k-1)(\text{scan}(|P| + |Q_h|))$ I/Os, by the induction hypothesis. Summing over all $\phi_k(|P|)$ levels, and adding the costs for answering left and right subqueries, we obtain the claim for $k > 1$. □

Choosing $k = 1$ for $d = 1$, and $k = \alpha(|P|)$ for $d > 1$, we obtain an $O(\text{sort}(|P| + |Q|))$ I/O algorithm for $d = 1$, and an $O\left(\frac{|P|\alpha^2(|P|)+|Q|\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ I/O algorithm for $d > 1$. Now partition P into contiguous subsets of size $\alpha(|P|)$ using $|P|/\alpha(|P|)$ splitters. Partition every query into left, middle, and right subqueries so that the left and right subqueries are maximized without spanning any splitter. For every query, answer the left and right subqueries by scanning the respective portion of P. This takes $O\left(\frac{|P|+|Q|}{DB}\alpha(|P|)\right)$ I/Os. Every middle subquery stretches between two splitters. Hence, we can represent every slab by a single point, thereby reducing the size of P to $|P|/\alpha(|P|)$, and we obtain the following theorem, applying Lem. 1 and Cor. 1.

Theorem 3. *It takes* $O\left(\text{sort}(|P| + |Q|) + \frac{(|P|+|Q|)\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ *I/Os and linear space to answer a set* Q *of range counting queries over a set* P *of points in* $\mathbb{R}^d$.

The *K-range minima problem* is to report the K points with minimum weight in each query range. Modifying the scan in Sect. 2.2 to maintain the K minima seen so far, it is easy to generalize the algorithm of this section to solve the K-range minima problem in $O\left(\text{sort}(|P| + |Q|) + \frac{(|P|+K|Q|)\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ I/Os, using $O\left(\frac{|P|+K|Q|}{B}\right)$ blocks of external memory.

3 Spanners for Point Sets

Computing K-th order θ-graphs. Our algorithm for constructing the K-th order θ-graph iterates over all cones $c \in C$ and computes the K closest points w.r.t. $dist_c$ in $c(p)$ for every point $p \in P$. Using an affine transformation, cone c can be transformed into the range $[0, +\infty) \times [0, +\infty) \times \cdots \times [0, +\infty)$, and reporting the K points closest to p in $c(p)$ translates into a K-range minima query for the modified cone $c'(p)$. Using Thm. 2, we obtain the following result.

Theorem 4. *Let P be a set of N points in $\mathbb{R}^d$, and $\theta < \pi$ be a constant angle. Then it takes $O\left(\text{sort}(N) + \frac{KN}{DB} \log^{d-1}_{M/B} \frac{N}{B}\right)$ I/Os and linear space to construct the K-th order θ-graph $G_{\theta,K}(P)$ of P.*

For fixed θ, the θ-graph has bounded out-degree, but not necessarily bounded in-degree. Using a two-phase approach, applying in both phases similar ideas to those presented in [8], we can compute in $O(\text{sort}(N))$ I/Os a spanner G of bounded in- and out-degree from a given θ-graph G'.[2]

Reporting paths in θ-graphs. A spanner path between two points s and t in a θ-graph can be computed as follows: For every point $p \in P$, determine the cone $c(p)$ containing t. Add the outgoing edge of p in cone $c(p)$ to a graph T. T is a tree with root t whose edges are directed towards the root. It follows directly from the arguments applied in the analysis of the spanning ratio of θ-graphs that the path from s to t in T is a spanner path. Hence, it is sufficient to report the path from s to t in T, which takes $O(\text{sort}(N))$ I/Os using time-forward processing [9].

4 *K*-Closest Pairs

The following theorem, which we prove in the full paper, can be used to find the K-closest pairs of a point set P in $O\big(\text{scan}\big(N\sqrt{K}\big)\big)$ I/Os, once an $O\big(\sqrt{K}\big)$-th order θ-graph has been constructed for P.

Theorem 5. *Let P be a set of points in $\mathbb{R}^d$, $0 < \theta < 2 \arccos \sqrt{4/5}$, $1 \le K \le n-1$, $K^* = \max(K, K^2/4)$, and $\{p, q\}$ be any of the K^* closest pairs in P. Then the K-th order θ-graph contains an edge between p and q.*

5 Spanners Among Polygonal Obstacles in the Plane

In this section, we prove the following theorem.

Theorem 6. *Given a set O of polygonal obstacles in the plane, with a total of N vertices, a linear size t-spanner for O can be computed in $O\left(\frac{N}{DB} \log_{M/(DB)} \frac{N}{DB}\right)$ I/Os using linear space.*

[2] The spanning ratio of G is $t = t' \cdot t'' \cdot \frac{1}{1-2\sin(B\pi/M)}$, where t' is the spanning ratio of G' and t'' is the spanning ratio of the single-sink spanners used in constructing G.

In [11], it is shown that the θ-graph $G_\theta(O)$ is such a t-spanner. Our algorithm to construct $G_\theta(O)$ follows the framework of [11]. However, we need to change the plane-sweep substantially, in order to perform it I/O-efficiently. For this purpose, we have to prove some new properties of conical Voronoi diagrams. The I/O-complexity of our algorithm becomes $O(\text{sort}(N))$ if the endpoint dominance problem [6] can be solved in $O(\text{sort}(N))$ I/Os.

The algorithm iterates over all cones $c \in C$ and computes the conical Voronoi diagram $CVD_c(O)$ (Fig. 1a) consisting of Voronoi regions V_x, $x \in P$. For each such region V_x and each point $p \in V_x$, x is the closest visible obstacle vertex in $c(p)$. Thus, for all obstacle vertices $y \in V_x$, the edge (y, x) is an edge of $G_\theta(O)$. Once $CVD_c(O)$ is computed, we add all such edges to the edge set of $G_\theta(O)$. In the full paper we show how to do this I/O-efficiently, so that Thm. 6 follows from the following result, which we prove in the rest of this section.

Theorem 7. *A representation of $CVD_c(O)$ as vertex and edge sets can be computed in $O\left(\frac{N}{DB}\log_{M/(DB)}\frac{N}{DB}\right)$ I/Os using $O(N/B)$ blocks of external memory.*

Assume that the coordinates have been transformed so that the cone axis of cone c points in positive y-direction. The construction of $CVD_c(O)$ uses a plane-sweep in this direction. Let c' be the reverse cone of c, i.e., the set of points p such that the apex of c' is contained in the cone $c(p)$. Then $V_x \subseteq c'(x)$. Denote the left and right boundaries of c' by h_l and h_r, respectively (Fig. 1b). Let H_l and H_r be two lines perpendicular to h_l and h_r, respectively, and directed upward. We denote the projections of a point p onto H_l and H_r by $H_l(p)$ and $H_r(p)$, respectively.

For the sake of simplicity, assume that the scene contains a dummy obstacle bounded from above by a horizontal line l_{bot} with y-coordinate $-\infty$. Now every cone $c'(x)$ becomes a triangle bounded by $h_l(x)$, $h_r(x)$, and a part of l_{bot}. As $V_x \subseteq c'(x)$, V_x is bounded from the left and right by $h_l(x)$ and $h_r(x)$ and from below by a polygonal chain consisting of parts of obstacle edges and/or parts of cone boundaries $h_l(y)$ and $h_r(y)$, for vertices $y \in P$ below x.

During the sweep, we maintain the invariant that the Voronoi regions V_y, for all vertices $y \in P$ below the sweep line ℓ, have been computed. Let P_ℓ be the set of vertices below ℓ and O_ℓ be the set of obstacles in O below or intersecting ℓ. Define the region $R_\ell = \bigcup_{y\in P_\ell} V_y \cup \bigcup_{o\in O_\ell} o$. As R_ℓ contains the dummy obstacle, it extends to infinity in both x-directions and in negative y-direction. The following lemma is proved in the full paper.

Lemma 2. *Region R_ℓ is connected.*

Lem. 2 implies that R_ℓ is bounded from above by a polygonal chain, even though it may contain holes. We call the upper boundary of R_ℓ the *horizon* of sweep line ℓ and denote it by $\mathcal{U}_\ell$ (Fig. 1a).

Let x be the current vertex on the sweep line ℓ, whose Voronoi region we want to compute. Let p_l be the first intersection point of $h_l(x)$ with $\mathcal{U}_\ell$ if $h_l(x)$ does not intersect the interior of an obstacle before intersecting $\mathcal{U}_\ell$. Otherwise, let $p_l = x$. We define a point p_r analogously with respect to $h_r(x)$. Then the edges $h_l^{\text{vor}}(x) = (x, p_l)$, $h_r^{\text{vor}}(x) = (x, p_r)$, and the part $\mathcal{U}_\ell(p_l, p_r)$ of $\mathcal{U}_\ell$ between

(a) (b)

Fig. 1. (a) A conical Voronoi diagram with the horizon $\mathcal{U}_\ell$ for sweep line ℓ shown in bold. Obstacles are shaded dark grey. Regions V_y for points y below the sweep line are shaded light grey. White regions below the horizon are holes in R_ℓ. (b) Illustrations of various definitions.

p_l and p_r bound a polygonal region R_x. Edge $h_l^{\text{vor}}(x)$ ($h_r^{\text{vor}}(x)$) is not defined if $p_l = x$ ($p_r = x$), and $R_x = \emptyset$ if $p_l = p_r = x$. The following lemma is shown in the full paper.

Lemma 3. *The region R_x defined as above is the Voronoi region of x. That is, $R_x = V_x$.*

Consider the region V_x. Lem. 3 implies that when the sweep line ℓ reaches vertex x, the whole boundary of V_x, except $h_l^{\text{vor}}(x)$ and $h_r^{\text{vor}}(x)$, has already been computed. To make the description of V_x complete, we need to compute $h_l^{\text{vor}}(x)$ and $h_r^{\text{vor}}(x)$. This can be done by performing two ray-shooting queries onto $\mathcal{U}_\ell$ in the directions of $h_l(x)$ and $h_r(x)$.

We answer ray-shooting queries in two stages. The first stage determines the points on obstacle boundaries hit by $h_l(x)$ and $h_r(x)$. The second stage determines whether $h_l(x)$ and $h_r(x)$ hit Voronoi edges $h_r^{\text{vor}}(y)$ and $h_l^{\text{vor}}(y')$ before hitting the respective obstacles.

The first stage is equivalent to solving the endpoint dominance problem for the set of obstacle vertices, which takes $O\left(\frac{N}{DB}\log_{M/(DB)}\frac{N}{DB}\right)$ I/Os [6]. Consider rays $h_l(x)$ and $h_r(x)$, for a vertex x on the boundary of an obstacle $o \in O$. Let $h_l^{\text{ext}}(x) = (x, y)$, where y is the first intersection of $h_l(x)$ with an obstacle. If $h_l(x) \setminus \{x\}$ intersects the interior of o before intersecting the boundary of o, $h_l^{\text{ext}}(x)$ is not defined. We define $h_r^{\text{ext}}(x)$ analogously. We say that segment $e = (x, y)$ *hits* segment $e' \subset \mathcal{U}_\ell$ if e and e' intersect in a point p and segment (x, p) does not intersect any other segment $e'' \subset \mathcal{U}_\ell$.

Observation 1. *The ray $h_l(x)$ ($h_r(x)$) hits a segment $h_r^{\text{vor}}(y)$ ($h_l^{\text{vor}}(y)$) if and only if segment $h_l^{\text{ext}}(x)$ ($h_r^{\text{ext}}(x)$) hits segment $h_r^{\text{vor}}(y)$ ($h_l^{\text{vor}}(y)$).*

If $h_l(x)$ does not hit an obstacle edge in $\mathcal{U}_\ell$, it hits a segment $h_r^{\text{vor}}(y)$ because segments $h_l^{\text{vor}}(y)$ and $h_l(y)$ are parallel. Analogously, $h_r(x)$ hits a segment $h_l^{\text{vor}}(y)$ if it does not hit an obstacle edge in $\mathcal{U}_\ell$. We show how to find the segment $h_r^{\text{vor}}(y)$ hit by $h_l^{\text{ext}}(x)$, and the segment $h_l^{\text{vor}}(y)$ hit by $h_r^{\text{ext}}(x)$, if any.

We do this in two separate plane sweeps. The first sweep finds segments $h_l^{\text{vor}}(y)$ hit by segments $h_r^{\text{ext}}(x)$, for all vertices $x \in P$. The second sweep finds all segments $h_r^{\text{vor}}(y)$ hit by segments $h_l^{\text{ext}}(x)$. As these two sweeps are similar, we only describe the first one. The difficulty is that segments $h_l^{\text{vor}}(y)$ are not known before the first sweep. They are only computed by the second sweep. However, the goal of the first sweep is not to compute segments $h_l^{\text{vor}}(y)$, but to determine whether a given segment $h_r^{\text{ext}}(x)$ hits such a segment $h_l^{\text{vor}}(y)$. The following lemma provides the tool to make this decision without knowing segments $h_l^{\text{vor}}(y)$ explicitly. For a given vertex x on sweep line ℓ, let the *hiding set* of $h_l^{\text{ext}}(x)$ be the set of endpoints z' of segments $h_l^{\text{ext}}(z')$ with $H_l(z') > H_l(x)$[3] and $y(z') < y(x)$. The following lemma is proved in the full paper.

Lemma 4. *Given a vertex x on sweep line ℓ, let $h_l^{\text{ext}}(x) = (x, y)$. Let z be the segment endpoint in the hiding set of $h_l^{\text{ext}}(x)$ such that $H_r(z)$ is maximized. If the hiding set of $h_l^{\text{ext}}(x)$ is empty, let $H_r(z) = -\infty$. Then there can be no vertex u above ℓ with $H_r(u) < \max\{H_r(y), H_r(z)\}$ such that $h_r^{\text{ext}}(u)$ hits $h_l^{\text{vor}}(x)$.*

Based on Lem. 4 we shorten segments $h_l^{\text{ext}}(x)$ to segments $h_l^{\text{vis}}(x)$ defined as follows: If point z in Lem. 4 exists and $H_r(z) > H_r(y)$, then $h_l^{\text{vis}}(x) = (x, y')$, where y' is the point on $h_l^{\text{ext}}(x)$ with $H_r(y') = H_r(z)$. Otherwise, $h_l^{\text{vis}}(x) = h_l^{\text{ext}}(x)$. Observe that $h_l^{\text{vis}}(x) \subseteq h_l^{\text{vor}}(x)$.

Corollary 2. *For a vertex $x \in P$, segment $h_r^{\text{ext}}(x)$ hits segment $h_l^{\text{vor}}(y)$ if and only if it hits $h_l^{\text{vis}}(y)$.*

Next we describe how to compute segments $h_l^{\text{vis}}(y)$ from the given set of segments $h_l^{\text{ext}}(y)$. Given segments $h_l^{\text{vis}}(y)$, we show how to compute the segment $h_l^{\text{vis}}(x)$ (and thus segment $h_l^{\text{vor}}(x)$) hit by $h_r^{\text{ext}}(u)$ (if any), for all $u \in P$. Once we know the intersection point of $h_r^{\text{ext}}(u)$ with this segment $h_l^{\text{vis}}(x)$, we can shorten $h_r^{\text{ext}}(u)$ to $h_r^{\text{vor}}(u)$.

Lem. 4 says that in order to compute $h_l^{\text{vis}}(x)$, we need to find the segment endpoint z below ℓ with $H_l(z) > H_l(x)$ and such that $H_r(z)$ is maximized over all such points. This is a partially unbounded range-maxima problem, which can be solved in $O(\text{sort}(N))$ I/Os, by Thm. 2.

Given segments $h_l^{\text{vis}}(x)$, ray-shooting queries for rays $h_r^{\text{ext}}(u)$ can be answered in $O(\text{sort}(N))$ I/Os using the distribution sweeping paradigm [13]: The slabs used in the recursive partition of the plane are parallel to h_r. The sweep, however, still proceeds in positive y-direction. For every slab σ, maintain the segment $\mu(\sigma)$ among all segments $h_l^{\text{vis}}(x)$ seen so far which maximizes $H_l(x)$ and completely spans σ. When the sweep reaches a new point u, determine the slab σ containing u, decide whether $h_r^{\text{ext}}(u)$ hits $\mu(\sigma)$ and update $\mu(\sigma')$ for all slabs σ' completely spanned by $h_l^{\text{vis}}(u)$.

Given segments $h_l^{\text{vor}}(x)$, $h_r^{\text{vor}}(x)$, and the obstacle edges, it now takes sorting and scanning to extract the vertex and edge sets of $CVD_c(O)$.

[3] Remember that H_l is directed to the left.

References

1. P. K. Agarwal, J. Erickson. Geometric range searching and its relatives. *Advances in Disc. and Comp. Geom.*, pp. 1–56. AMS, 1999.
2. L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. S. Vitter. Theory and practice of I/O-efficient algorithms for multidimensional batched searching problems. *Proc. SODA*, 1998.
3. L. Arge, V. Samoladas, J. S. Vitter. On two-dimensional indexability and optimal range search indexing. *Proc. PODS'99*, 1999.
4. L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. *Proc. WADS*, pp. 334–345, 1995.
5. L. Arge, P. B. Miltersen. On showing lower bounds for external-memory computational geometry problems. In J. Abello and J. S. Vitter (eds.), *External Memory Algorithms and Visualization*. AMS, 1999.
6. L. Arge, D. E. Vengroff, J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. *Proc. ESA*, pp. 295–310, 1995.
7. L. Arge, J. S. Vitter. Optimal dynamic interval management in external memory. *Proc. FOCS*, pp. 560–569, 1996.
8. S. Arya, G. Das, D. M. Mount, J. S. Salowe, M. Smid. Euclidean spanners: Short, thin, and lanky. *Proc. STOC*, pp. 489–498, 1995.
9. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, J. S. Vitter. External-memory graph algorithms. *Proc. SODA*, 1995.
10. P. B. Callahan, S. R. Kosaraju. A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Proc. STOC*, pp. 546–556, 1992.
11. K. L. Clarkson. Approximation algorithms for shortest path motion planning. *Proc. STOC*, pp. 56–65, 1987.
12. M. Fischer, T. Lukovszki, M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. *Proc. ESA*, pp. 163–174, 1998.
13. M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, J. S. Vitter. External-memory computational geometry. *Proc. FOCS*, 1993.
14. S. Govindarajan, T. Lukovszki, A. Maheshwari, N. Zeh. I/O-efficient well-separated pair decomposition and its applications. *Proc. ESA*, pp. 220–231, 2000.
15. Y. Hassin, D. Peleg. Sparse communication networks and efficient routing in the plane. *Proc. PODC*, 2000.
16. J. M. Keil, C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.
17. C. Levcopoulos, G. Narasimhan, M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. *Proc. STOC*, pp. 186–195, 1998.
18. T. Lukovszki. New results on fault tolerant geometric spanners. *Proc. WADS*, pp. 193–204, 1999.
19. J. Ruppert, R. Seidel. Approximating the d-dimensional complete Euclidean graph. *Proc. CCCG*, pp. 207–210, 1991.
20. D. E. Vengroff, J. S. Vitter. Efficient 3-D range searching in external memory. *Proc. STOC*, 1996.
21. J. S. Vitter. External memory algorithms. *Proc. PODS*, pp. 119–128, 1998.
22. J. S. Vitter, E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.
23. A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comp.*, 11:721–736, 1982.

Beyond Message Sequence Graphs

P. Madhusudan[1] and B. Meenakshi[2]

[1] Chennai Mathematical Institute, Chennai, India
madhu@imsc.ernet.in
[2] The Institute of Mathematical Sciences, Chennai, India
bmeena@imsc.ernet.in

Abstract. We study the model-checking problem for classes of message sequence charts (MSCs) defined by two extensions of message sequence graphs (MSGs). These classes subsume the class of regular MSC languages. We show that the model checking problem for these extended message sequence graphs against monadic second-order specifications is decidable. Moreover, we present two ways to model-check the extended classes — one extends the proof for MSGs while the other extends the proof for regular MSC languages.

1 Introduction

Message sequence charts (MSCs) is an ITU-standardized notation widely used to capture system requirements in the early stages of design of communication protocols [ITU97,RGG96]. The clear graphical layout of an MSC describes how the components of a distributed system communicate by exchanging messages. In its simplest form, an MSC depicts the exchange of messages between processes of a distributed system and corresponds to a single partially-ordered execution of the system. A standard way to specify a collection of scenarios is by using *message sequence graphs* (MSGs) (also known as high-level MSCs). MSGs allow MSCs to be combined using the operations of choice, concatenation and repetition.

In [AY99], the authors study the (linearization) model-checking problem for message sequence graphs. One is given an MSG and a regular set of sequences of actions as the specification and asked whether for every MSC represented by the MSG, all the linearizations of the MSC are included in the specification. This problem is shown to be undecidable, and the authors go on to define a restricted class of MSGs for which the problem is decidable.

The model-checking problem of MSGs against monadic second-order (MSO) formulas, which are interpreted directly on MSCs and not on its linearizations, was studied in [Mad01]. It was shown that the model-checking problem was decidable without any restriction on the MSGs. Independently, Peled has shown that model-checking a fragment of TLC (a temporal logic on partial-orders) is solvable for unrestricted MSGs [Pel00]. These results suggest that structural logics may be more amenable to verification than those based on linearizations.

In [HMKT00b], the authors study another class of MSCs called *regular MSC-languages*. It turns out that using a proof in [HMKT00b], one can show that the

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 256–267, 2001.
© Springer-Verlag Berlin Heidelberg 2001

model-checking problem for regular MSC languages against MSO specifications is decidable as well. However, the class of MSC-languages defined by MSGs and the class of regular MSC-languages are incomparable — there are MSC-languages that are definable in one but not in the other [HMKT00a].

Thus, though the model-checking problem for MSO formulas is decidable for MSC-languages presented as MSGs as well as for regular MSC languages, the proof methodologies seem very different. While the former works by viewing concatenated MSCs as meta-strings where each letter represents an MSC, the latter works using the set of all linearizations of the MSCs.

We unify the proof mechanisms in the above works by considering a larger class of MSCs through the formalism of *compositional Message Sequence Graphs* (CMSGs) introduced in [GMP01]. These are similar to MSGs, except that one is allowed to have *unmatched* send-events (send-events whose corresponding receive-events do not occur in the current atomic MSC) while forming an MSC, which will be matched with other receive-events in other MSCs. In [GMP01] it was shown that this is a strictly more expressive formalism than that of an MSG, and that it allows modelling of interesting scenarios such as the behaviour of the alternating-bit protocol.

In terms of linearizations, a regular MSC language is one such that *all* the linearizations of all MSCs in it is a regular set of sequences [HMKT00b]. We can relax this condition, and represent an MSC-language by a subset L of linearizations of MSCs in the MSC-language such that L has *at least one* linearization for each MSC in the language. When L is regular, we say that the MSC-language has a *regular representative linearization*. It then turns out that the class of CMSG definable languages is precisely the class of MSC-languages that have a regular representative linearization. Consequently, this class of languages subsumes the class of MSG-definable languages as well as regular MSC-languages.

We show that the model-checking problem for CMSGs against MSO specifications is decidable. More importantly, we identify two ways of solving this problem — one extending the method used to model-check MSGs in [Mad01] and the other technique implicit in [HMKT99] to model-check regular MSC-languages.

We then go on to define and study a formalism more general and flexible than a CMSG, called *extended CMSGs* (XCMSGs). In this setting, apart from unmatched send-events, one is also allowed to have unmatched receive-events when traversing a path in the MSG. We show that the notion analogous to linearizations in this setting is that of *semi-linearizations* — an ordering of events which respects the local total orders of the MSC but *does not* necessarily respect the causal order. We show then theorems analogous to the above results — equivalence between MSC-languages defined by XCMSGs and those that have regular representative semi-linearizations and how the two proofs of decidability of the model-checking problem extend to this class as well.

We believe our results will help understand the nuances involved in model-checking classes of MSC-languages, which borders on the line of undecidability. On the theoretical side, checking MSO properties of infinite graphs and infinite

classes of graphs is an interesting area of research [Tho97]. Our proofs using linearizations and semi-linearizations seem to be suggest a new technique to tackle this problem for certain classes of graphs.

Due to space constraints, we present only the gist of proofs — more details and related results (like extensions to infinite MSCs, etc.) can be found in [MM01].

2 Preliminaries

Let us fix a finite set of processes $\mathcal{P}$ for the rest of the paper and let p, q, r range over $\mathcal{P}$. Let us also fix a finite set of messages Γ_m and a finite set of local actions Γ_l. Let $\Sigma_p = \{(p!q, a), (p?q, a), (p, l) \mid p, q \in \mathcal{P}, p \neq q, a \in \Gamma_m, l \in \Gamma_l\}$ denote the set of actions that p participates in. The action $(p!q, a)$ should be read as "p sends the message a to q" while the action $(p?q, a)$ means "p receives the message a from q". (p, l) represents p doing a local action l. Let $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$ denote the set of all actions.

Definition 1. *A* message sequence chart *(MSC) over $\mathcal{P}$ is a tuple $m = (E, \{\leq_p\}_{p \in \mathcal{P}}, \lambda, \eta)$ where*

- *E is a finite set of events.*
- *$\lambda : E \to \Sigma$ is the labelling function which identifies for each event an action. Let $E_p = \{e \in E \mid \lambda(e) \in \Sigma_p\}$ denote the set of events of E which p participates in. Also, let $S_E = \{e \in E \mid \lambda(e) = (p!q, a)$ for some $p, q \in \mathcal{P}, a \in \Gamma_m\}$ denote the set of send-events and $R_E = \{e \in E \mid \lambda(e) = (p?q, a)$ for some $p, q \in \mathcal{P}, a \in \Gamma_m\}$ denote the set of receive-events of E.*
- *$\eta : S_E \to R_E$ is the matching function which associates with each send-event, its corresponding receive-event. We require η to be a bijection and for every $e, e' \in E$, if $\eta(e) = e'$, then $\lambda(e) = (p!q, a)$ and $\lambda(e') = (q?p, a)$ for some $p, q \in \mathcal{P}, a \in \Gamma_m$.*
- *$\leq_p$ is a total order on E_p for each $p \in \mathcal{P}$.*
- *Let $\widehat{\leq} = (\bigcup_{p \in \mathcal{P}} \leq_p) \cup \{(e, e') \mid e, e' \in E$ and $\eta(e) = e'\}$. Let $\leq = (\widehat{\leq})^*$ be the transitive closure of $\widehat{\leq}$. Then $\leq$ denotes the* causal ordering *of events in the MSC and we require it to be a partial order on E.*
- (non-degeneracy) *m is non-degenerate ([AEY00]) in the sense that two identical messages sent by a process are not received in the reverse order: i.e. if there are events e_1, e_2, e_1', e_2' such that $\lambda(e_1) = \lambda(e_2) = (p!q, a)$, $\eta(e_1) = e_1'$, $\eta(e_2) = e_2'$ and $e_1 <_p e_2$, then it must be the case that $e_1' <_q e_2'$.* □

An *event-linearization* of an MSC m is a linear order on E which extends $\leq$, the causal order of m. We represent event-linearizations as sequences over E: a sequence $e_1, \ldots, e_k$ represents the event-linearization $\sqsubseteq$ given by $e_i \sqsubseteq e_j$ iff $i \leq j$.

A *linearization* of an MSC m is any possible sequence of actions which the MSC describes. Formally, $lin(m)$, the set of linearizations of m, is the set of

sequences $\lambda(e_1), \ldots, \lambda(e_k)$ where $e_1, \ldots, e_k$ is an event-linearization of m. Linearizations are represented as words in Σ^*.

Note that while an MSC defines a nonempty set of linearizations, one can associate a unique MSC (up to isomorphism) with a given linearization by defining an appropriate set of events and matching the sends and receives using the non-degeneracy condition for an MSC. For a linearization $w \in \Sigma^*$, we can define the MSC corresponding to w as $(E, \{\leq_p\}_{p \in \mathcal{P}}, \lambda, \eta)$, where E is the set of all non-epsilon prefixes of w, $\lambda(x.r) = r$, where $x.r \in E$, $x \in \Sigma^*$, $r \in \Sigma$. $x.r \leq_p y.r'$ iff $x.r$ is a prefix of $y.r'$ and $r, r' \in \Sigma_p$. $\eta(x.r) = y.r'$ iff $r = (p!q, a)$, $r' = (q?p, a)$ and the number of occurrences of r in x is equal to the number of occurrences of r' in y.

For any $w \in \Sigma^*$, if for every $p, q \in \mathcal{P}$, $a \in \Gamma_m$, and every prefix y of w, the number of occurrences of $(p!q, a)$ in y is at least the number of occurrences of $(q?p, a)$ in y, and the number of occurrences of $(p!q, a)$ in w is the same as the number of occurrences of $(q?p, a)$ in w, then one can associate an MSC m for which w is a linearization. We call such words *well-formed* words. For such a $w \in \Sigma^*$, let $msc(w)$ denote the MSC corresponding to w.

We can now represent a collection of MSCs through a set of *representative linearizations*. Let $L \subseteq \Sigma^*$ be a set of well-formed words. Then L represents the class of MSCs $msc(L) = \{msc(w) \mid w \in L\}$. Note that we do not require L to have *all* the linearizations of the MSCs it represents (i.e. L need not be equal to $lin(msc(L))$), as required for *regular MSC languages* [HMKT00b]. We say that L is a representative linearization of a collection of MSCs if L represents this collection (up to isomorphism).

Another way to define a collection of MSCs is through the notion of a *Message Sequence Graph* (MSG). This is a mechanism whereby one takes a finite set of MSCs and defines regular ways of combining them using choice, concatenation and repetition. In this paper, we will consider a more general notion of MSGs, called *compositional* MSGs, or CMSGs.

Let us first define a compositional MSC (CMSC). A CMSC is basically an MSC which, in addition to the normal messages in an MSC, can also have unmatched send and receive events — these will be matched up later using corresponding unmatched receive and send events respectively, when the CMSC is composed with another.

Definition 2. *A* compositional message sequence chart *(CMSC) over $\mathcal{P}$ is a tuple $\hat{m} = (E, \{\leq_p\}_{p \in \mathcal{P}}, \lambda, \eta)$ where E, λ and $\leq_p$ are as in an MSC and $\eta : S_E \rightharpoonup R_E$ is a* partial *matching function which associates with some send-events, their corresponding receive-events. We require η to be injective and for every $e, e' \in E$, if $\eta(e) = e'$, then $\lambda(e) = (p!q, a)$ and $\lambda(e') = (q?p, a)$ for some $p, q \in \mathcal{P}, a \in \Gamma_m$. Using the events on which η is defined, we can define the causal order, as in an MSC. We require this to be a partial order. We also require $\hat{m}$ to be non-degenerate (with respect to the messages matched by η).* □

We say that $e \in S_E$ is an *unmatched send-event* if η is not defined on e and $e \in R_E$ is an *unmatched receive-event* if $e \notin \eta(S_E)$. For a CMSC $\hat{m}$ and $(p!q, a) \in$

Σ, let $\#^{unm}_{(p!q,a)}(\hat{m}) = |\{e \in E \mid \lambda(e) = (p!q, a) \text{ and } \eta(e) \text{ is not defined } \}|$ denote the number of unmatched send-events of the type $(p!q, a)$.

Concatenation of two CMSCs is done by (asynchronously) concatenating the events of each process, and by matching up some unmatched send-events of the first CMSC with corresponding unmatched receive-events of the second.

Definition 3. *Let $\hat{m}_i = (E_i, \{\leq^i_p\}_{p\in\mathcal{P}}, \lambda_i, \eta_i)$, $i = 1, 2$ be two CMSCs, with $E_1 \cap E_2 = \emptyset$. Also, let $\hat{m}_1$ have no unmatched receives. Let $\hat{m} = (E, \{\leq_p\}_{p\in\mathcal{P}}, \lambda, \eta)$ where $E = E_1 \cup E_2$; $\forall e \in E_1, \lambda(e) = \lambda_1(e)$ and $\forall e \in E_2, \lambda(e) = \lambda_2(e)$; $\leq_p = \leq^1_p \cup \leq^2_p \cup \{(e_1, e_2) \mid e_1 \in E_1 \cap E_p, e_2 \in E_2 \cap E_p\}$ and η is defined as follows:*

(C1) *If $e \in S_{E_i}$ and $\eta_i(e)$ is defined, where $i \in \{1, 2\}$, then $\eta(e) = \eta_i(e)$.*
(C2) *Let $e \in S_{E_1}$, η_1 be undefined on e, and $\lambda(e) = (p!q, a)$. If there is an event $e' \in R_{E_2}$ such that η_2^{-1} is undefined on e' (i.e. $e' \notin \eta_2(S_{E_2})$), $\lambda(e') = (q?p, a)$ and $|\{f \in E \mid f \leq_p e, \lambda(f) = (p!q, a)\}| = |\{f' \in E \mid f' \leq_p e', \lambda(f') = (q?p, a)\}|$, then we set $\eta(e) = e'$. Clearly, if such an e' exists, it is unique.*
(C3) *η is undefined on all other events of S_E.*

If $\hat{m}$ is non-degenerate, then it is easy to see that $\hat{m}$ is a CMSC, and we define the concatenation *of $\hat{m}_1$ and $\hat{m}_2$, denoted by $\hat{m}_1 \cdot \hat{m}_2$, to be $\hat{m}$. Otherwise, concatenation is not defined.* □

When we write a series of concatenations $\hat{m}_1 \cdot \hat{m}_2 \cdot \hat{m}_3 \cdot \ldots$ we always mean a left-to-right application, i.e. the term $((\hat{m}_1 \cdot \hat{m}_2) \cdot \hat{m}_3) \ldots$. Note that concatenation is sensitive to the order in which it is made.

Let us now fix some notation for describing automata. Let A be a finite alphabet. A deterministic finite automaton (DFA) over A is a tuple $\mathcal{A} = (S, s_{in}, \delta, F)$ where S is a finite set of states, $s_{in} \in S$ is the initial state, $\delta : S \times A \to S$ is the transition function and $F \subseteq S$ is the set of final states. δ can be extended to $\delta' : S \times A^* \to S$ by defining $\delta'(s, \epsilon) = s$ and $\delta'(s, x.d) = \delta(\delta'(s, x), d)$ where $x \in A^*$ and $d \in A$. We say that a word $x \in A^*$ is accepted by $\mathcal{A}$ if $\delta'(s_{in}, x) \in F$. The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words in A^* accepted by $\mathcal{A}$.

We are now ready to define CMSGs. Let us fix a finite set of CMSCs M — we will call these *atomic* CMSCs. We say that a sequence of CMSCs $\hat{m}_1, \hat{m}_2, \ldots \hat{m}_n$ from M is *well-defined* iff the concatenation of the CMSCs in this order is defined. We say that it is *complete* if the concatenated CMSC has no unmatched send or receive events — i.e. it is an MSC.

Let Π be a finite alphabet and $h : \Pi \to M$ be a bijection. Given a word $x = d_0 \ldots d_n \in \Pi^*$, if $h(d_0), \ldots h(d_n)$ is well-defined, then x defines the CMSC $cmsc(x) = h(d_0) \cdot h(d_1) \ldots h(d_n)$. We say that x *is well-defined* if $h(d_0), \ldots, h(d_n)$ is well-defined and say x *is complete* if $cmsc(x)$ is an MSC.

Definition 4. *A* compositional message sequence graph *(CMSG) is a tuple $G = (\Pi, M, h, \mathcal{A})$ where Π is a finite alphabet, M is a finite set of CMSCs, $h : \Pi \to M$ is a bijection, and $\mathcal{A}$ is a DFA over Π. We require that every $x \in L(\mathcal{A})$ is well-defined and complete. The CMSG represents the set of MSCs $msc(G) = \{cmsc(x) \mid x \in L(\mathcal{A})\}$.* □

For example, consider the CMSG in Figure 1. It depicts a producer-consumer example where p sends messages continuously to c. At some point, c sends the "abort" signal to p requesting it to stop sending messages but this message takes an arbitrarily long time to reach p. The figure shows a typical behaviour and a CMSG which represents such scenarios.

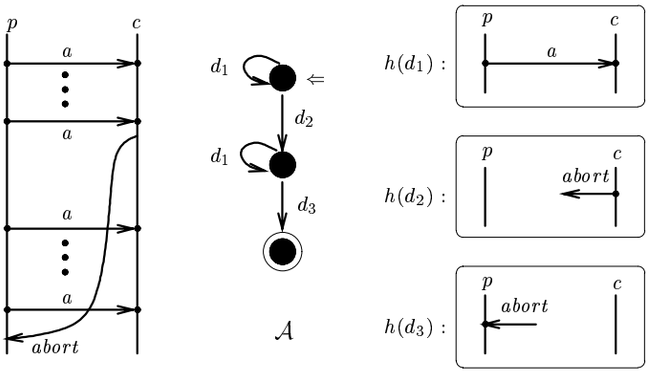


Fig. 1. A compositional message sequence graph

It turns out that given a structure $G = (\Pi, M, h, \mathcal{A})$, we can decide if it is a CMSG.

One way to simplify CMSGs is to partition Γ_m into two sets Γ_m^1 and Γ_m^2 and always use Γ_m^1-events for unmatched sends and receives, while using Γ_m^2 for matched sends and receives, in all the atomic CMSCs (as done in [GMP01]). Such a condition would ensure that when concatenating CMSCs, the resulting CMSC will never be degenerate. Also, one can show that any CMSG can be converted to an equivalent one over such restricted atomic CMSCs.

MSGs are just like CMSGs except that the atomic CMSCs are in fact MSCs. The class of CMSGs clearly extend that of MSGs — in fact, they are strictly more powerful. It can be shown that the behaviour illustrated in Figure 1 cannot be represented by MSGs nor is a regular MSC-language (as defined in [HMKT00b]).

We use monadic second order logic (MSO) to describe properties of MSCs. This logic is defined as follows: We have at our disposal a countable number of first-order variables $\{x, y, z, \ldots\}$ and second-order variables $\{X, Y, Z, \ldots\}$ — the former will be interpreted over events while the latter over subsets of events of an MSC. The atomic formulas are of the kind $(x \rightarrow y)$ (meaning that x is a send-event whose corresponding receive-event is y), $(x \leq_p y)$ (meaning x and y are events of p and x is before y in the total order of process p), $Q_r(x)$, where $r \in \Sigma$ (meaning the label of the event x is r), and $(x \in X)$. Other formulas are formed using the boolean connectives $\vee$ and $\neg$ and using quantification over first-order and second-order variables.

This logic is a powerful structural logic and can express many interesting properties of MSCs. It can be shown, for example, that the temporal logic TLC over MSCs considered in [Pel00] can be embedded into MSO.

Let φ be a formula and $m = (E, \{\leq_p\}_{p\in\mathcal{P}}, \lambda, \eta)$ be an MSC. An interpretation of a set of first-order and second-order variables V is a function I that assigns to each first-order variable an event in E, and assigns a set of events of E to each second-order variable. Semantics of when m satisfies φ under an interpretation I is defined in the obvious way and leads to the definition of when m satisfies a sentence φ.

3 CMSGs and Regular Representative Linearizations

We now show that the class of MSC-languages represented by CMSGs is the same as that definable using a regular set of representative linearizations.

Lemma 1. *Let $\mathcal{L}$ be a collection of MSCs over $\mathcal{P}$. There exists a CMSG $G = (\Pi, M, h, \mathcal{A})$ that represents $\mathcal{L}$ iff there exists a regular set $L \subseteq \Sigma^*$ of well-formed words which is a representative linearization of $\mathcal{L}$.*

Proof: Let $G = (\Pi, M, h, \mathcal{A})$ be a CMSG such that $msc(G) = \mathcal{L}$. For every $\hat{m} \in M$, let us fix one linearization of $\hat{m}$, $\widehat{lin}(\hat{m})$. Let $L = \{\widehat{lin}(h(d_1)) \ldots \widehat{lin}(h(d_n)) \mid d_1 \ldots d_n \in L(\mathcal{A})\}$. It is easy to see that L is a regular representative linearization of $\mathcal{L}$. The other direction follows from [GMP01]. Let $L \subseteq \Sigma^*$ be a representative linearization of $\mathcal{L}$, and L be a regular language accepted by a DFA $\mathcal{B} = (S, s_{in}, \delta, F)$ over Σ. For every $r \in \Sigma_q$, let M have a CMSC with a *single event* r labelled r. One can now use $\mathcal{B}$ itself to define a CMSG which has L as a representative linearization. □

Let us now turn to the model checking problem for CMSGs: Given a CMSG G (or a regular representative linearization L), and a MSO sentence φ, do all the MSCs represented by G satisfy φ? We show that this problem is decidable. We exhibit two proofs of this theorem, one which extends the proof of decidability of model-checking MSGs [Mad01] and the other which extends the proof of decidability of model-checking for regular MSC-languages [HMKT00b].

A CMSC $m = (E, \{\leq_p\}_{p\in\mathcal{P}}, \lambda, \eta)$ is said to be b-memory bounded, where $b \in \mathbb{N}$, if the number of unmatched sends of any type is at most b, i.e., $\#^{unm}_{(p!q,a)}(m) \leq b$. A sequence of CMSCs $m_1, m_2, \ldots, m_k$ is b-memory bounded if for all its prefixes $m_1, m_2, \ldots, m_i$, the CMSC $m_1 \cdot m_2 \ldots \cdot m_i$ is b-memory bounded, where $i \leq k$. A CMSG $G = (\Pi, M, h, \mathcal{A})$ is b-memory bounded if for every $d_1 d_2 \ldots d_n \in L(\mathcal{A})$, the sequence of CMSCs $h(d_1), h(d_2), \ldots, h(d_n)$ is b-memory bounded. The fact that a CMSG defines a collection of MSCs forces it to be b-memory bounded:

Lemma 2. *For every CMSG G, there exists $b \in \mathbb{N}$, which is computable, such that G is b-memory bounded.*

Proof: Let $G = (\Pi, M, h, \mathcal{A})$ be a CMSG and let $\mathcal{A} = (S, s_{in}, \delta, F)$. For any state $s \in S$ which is reachable from the initial state and from which the final

state can be reached, one can show that all CMSCs defined on a path from the initial state to s must have the same number of unmatched receives (from the fact that all accepting paths define MSCs). It then follows that if b is the maximum number of unmatched receives at any such state, then G is b-memory bounded. □

The following lemma will be used to show that the model checking problem for CMSGs is decidable.

Lemma 3. *Let Π be a finite alphabet, M be a set of CMSCs and $h : \Pi \to M$ be a bijection. Let φ be an MSO sentence and $b \in \mathbb{N}$. Then the collection of words $w = d_1 \dots d_n \in \Pi^*$ such that $h(d_1), \dots h(d_n)$ is well-defined, complete and b-memory bounded, and $cmsc(w) \models \varphi$, is a regular subset of Π^*.*

Proof: The proof follows the corresponding proof for MSCs in [Mad01]. Let φ be an MSO formula with free second-order variables $V_\varphi = \{X_1, \dots X_k\}$ (assume for now that we have only second-order variables). Then let $\Pi_\varphi = \{(d, I) \mid d \in \Pi, I : V_\varphi \to 2^E,$ where E is the set of events of $h(d)\}$. Each letter in Π_φ encodes along with a letter $d \in \Pi$, an interpretation of the free variables over the events of the CMSC corresponding to d. The idea is to construct an automaton $\mathcal{A}_\varphi$ which will accept a word $(d_1, I_1) \dots (d_n, I_n)$ iff the sequence $h(d_1), \dots h(d_n)$ is well-defined, complete and b-memory bounded, and the MSC $m = cmsc(d_1 \dots d_n)$ under the combined interpretation defined by $I_1, \dots I_n$ on the events of m, satisfies φ. This is done inductively on the structure of the formula.

First, the set of all words $d_1 \dots d_n \in \Pi^*$ such that $h(d_1), \dots h(d_n)$ is a well-defined and complete b-memory bounded sequence, is regular. We run an automaton accepting this in parallel with the automaton we construct.

The atomic formula $x \to y$ is the hardest to handle. The automaton will check the interpretation indeed maps it to singleton sets — assume it does. The automaton now has to check if y is the corresponding receive-event of x. Since the sequence we are reading is b-memory bounded, we will know the number of sends of the kind $(p!q, a)$ which are pending when x happens — let this be k (k is bounded by the sum of b and the maximum number of events in an atomic CMSC). The event y is the corresponding receive-event of x, if the number of receive-events of the kind $(q?p, a)$ between x and y is k. For each atomic CMSC, we know a priori the number of receive-events of this kind it has. Since the number of atomic CMSCs between the atomic CMSCs which x and y belong to that have at least one receive-event of this kind is bounded, we can construct with some bookkeeping an automaton which checks this property.

Other atomic formulas are easy to handle; formulas obtained by disjunction, negation and existential quantification can be handled by using the fact that automata are closed under union, complement and projection respectively. □

Theorem 1. *The model checking problem for CMSGs against MSO-formulas is decidable.*

Proof: Let $G = (\Pi, M, h, \mathcal{A})$ be a CMSG and φ be an MSO-formula. From Lemma 2, it follows that there exists $b \in \mathbb{N}$ (which can be computed) such that

G is b-memory bounded. Construct using Lemma 3, an automaton $\mathcal{A}^b_\varphi$ over Π^* that accepts a word $w = d_1 \dots d_n$ iff $h(d_1), \dots h(d_n)$ is well-defined, complete and b-memory bounded, and $cmsc(w) \models \varphi$. Clearly, all MSCs in $msc(G)$ satisfy φ iff $L(\mathcal{A}) \subseteq L(\mathcal{A}^b_\varphi)$, which can be checked. □

Let us now turn to linearizations and use them to solve the model checking problem. Let $w \in \Sigma^*$ be a well-formed word and $b \in \mathbb{N}$. w is said to be *b-bounded* if for every prefix x of w, the difference between the number of send-events in x of the type $(p!q, a)$ and the number of receive-events of the same type is at most b. A language is said to be b-bounded if each word in it is b-bounded.

Lemma 4. *Given an MSO-formula φ and $b \in \mathbb{N}$, the set of all well-formed words $w \in \Sigma^*$ such that w is b-bounded and $msc(w) \models \varphi$ is regular.*

Proof: From φ we construct, using the technique in [HMKT99, Lemma 4.1], an MSO formula φ^b_{str} over finite words in Σ^* such that for $w \in \Sigma^*$, $w \models \varphi^b_{str}$ iff w well-formed, b-bounded and $msc(w) \models \varphi$. The required conclusion follows since the strings described by MSO formulae on words form a regular set [Tho90]. The formula φ^b_{str} is constructed so that it matches the send and receive events of the MSC corresponding to the word, and dynamically interprets φ using this matching and the local total orders. □

We can now give another proof of the result that model checking CMSGs against MSO specifications is decidable using linearizations.

Alternative proof of Theorem 1: In view of Lemma 1, let the CMSG be presented as a regular representative linearization L via a DFA $\mathcal{B}$ accepting L. Since any regular language of well-formed words is b-bounded (see [HMKT00b]), where b can be computed, we can find a b such that L is b-bounded. Construct using Lemma 4, an automaton $\mathcal{B}'$ that accepts all b-bounded words which represent MSCs that satisfy φ. Clearly, all the MSCs represented by L satisfy φ iff $L(\mathcal{B}) \subseteq L(\mathcal{B}')$, which is decidable. □

Another important (albeit simple) consequence of dealing with representative linearizations is that it allows us to show decidability of the *linearization model checking problem* [AY99] for *trace-closed* or linearization-closed specifications. The linearization model checking problem for CMSGs is: Given a CMSG G and a regular language L over Σ, is $lin(msc(G)) \subseteq L$? This problem is known to be undecidable even for MSGs [AY99]. However, if L is linearization-closed, we can effectively solve the problem: Using Lemma 1, construct L' which is a regular representative linearization of $msc(G)$. It is now easy to verify that all linearizations of MSCs represented by G belong to L iff $L' \subseteq L$.

Also, the above procedure works in time polynomial in the sizes of the given CMSG and the specification automaton. The ease of the above model-checking problem raises the hope for defining temporal logics whose models are trace-closed (akin to LTrL in the setup of Mazurkiewicz traces [TW97]), as these logics may yield a fast verification procedure. The logic m-LTL introduced in [MR00] and the logic TLC for MSCs [Pel00] can be seen as starting points for this endeavour.

4 Extended CMSGs

In the definition of concatenation of two CMSCs $\hat{m}_1$ and $\hat{m}_2$ (Definition 3), we required that $\hat{m}_1$ have no unmatched receives. We can drop this requirement and define concatenation so that it matches unmatched receive-events of $\hat{m}_1$ with corresponding unmatched send-events in $\hat{m}_2$ (if any). This relaxed definition is the same as that of Definition 3 except that condition (C2) is replaced by:

(C2′) Let $e \in S_E$, η_1 and η_2 be undefined on e, and $\lambda(e) = (p!q, a)$. If there is an event $e' \in R_E$ such that η_1 and η_2 are undefined on e', $\lambda(e') = (q?p, a)$ and $|\{f \in E \mid f \leq_p e, \lambda(f) = (p!q, a)\}| = |\{f' \in E \mid f' \leq_p e', \lambda(f') = (q?p, a)\}|$, then we set $\eta(e) = e'$.

Again, this concatenation is defined only if the resulting structure is a CMSC (i.e. the resulting causal order is a partial-order and is non-degenerate).

We can now define *extended CMSGs* (XCMSG) which are exactly like CMSGs (see Definition 4) except that the MSCs represented by them are defined using the relaxed notion of concatenation above. As in a CMSG, we require that the accepting paths of the XCMSG define MSCs.

Again, given a structure $G = (\Pi, M, h, \mathcal{A})$, one can decide if it is a XCMSG. Also, the class of MSCs definable in this fashion is strictly larger than the class defined by CMSGs. For example, Figure 2 illustrates an XCMSG and a typical MSC represented by it. The figure describes a scenario where three processes p, q and r exchange messages. The channel from p to r is a slow data channel while the channels from p to q and q to r are fast. Consequently, the messages b followed by c overtake an arbitrary number of messages a from p to r. It is not difficult to see that there is no equivalent CMSG representation for it.

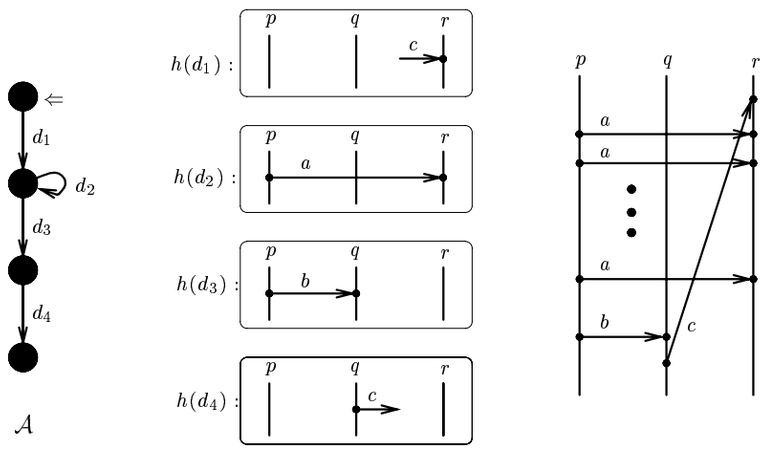


Fig. 2. An extended CMSG

However, XCMSGs are amenable to model checking against MSO specifications. Analogous to the definition of b-memory boundedness for CMSCs, one can define the notion of (b', b)-*memory bounded* CMSC, where $b, b' \in \mathbb{N}$. A CMSC m is (b', b)-memory bounded if the number of unmatched sends of any type is at most b (i.e., it is b-memory bounded) and the number of unmatched receives of any type is bounded by b'. We can extend this notion to sequences of CMSCs and to XCMSGs. It then turns out that any XCMSG is (b', b)-memory bounded for some $b, b' \in \mathbb{N}$ and we have a result analogous to Lemma 3:

Lemma 5. *Let Π be a finite alphabet, M be a set of CMSCs and $h : \Pi \to M$ be a bijection. Let φ be an MSO formula and $b, b' \in \mathbb{N}$. Then the collection of words $w = d_1 \dots d_n \in \Pi^*$ such that $h(d_1), \dots h(d_n)$ is well-defined, complete and (b', b)-memory bounded, and $cmsc(w) \models \varphi$, is a regular subset of Π^*.* □

We then have

Theorem 2. *The model checking problem for XCMSGs against MSO specifications is decidable.* □

The example in Figure 2 also illustrates the fact that there is no regular representative linearization of the MSCs represented by the given XCMSG. It turns out that the notion analogous to linearizations in this setting is that of a *semi-linearization*. For an MSC $m = (E, \{\leq_p\}_{\{p \in \mathcal{P}\}}, \lambda, \eta)$, an *event semi-linearization* of m is a linear order on E which extends the orders $\leq_p$ for each $p \in \mathcal{P}$. Note that the order *need not* extend the causal ordering $\leq$ — a receive-event could be ordered before its corresponding send-event. The semi-linearizations of an MSC m is the set of sequences $\lambda(e_1), \dots, \lambda(e_k)$ where $e_1, \dots, e_k$ is an event semi-linearization of m.

Again, while an MSC defines a non-empty set of semi-linearizations, a semi-linearization w corresponds to a unique MSC, denoted by $msc(w)$. For a word $w \in \Sigma^*$, we say w is *semi well-formed* if there is an MSC for which w is a semi-linearization We can now represent a collection of MSCs through a set of representative semi-linearizations. The following is analogous to Lemma 1.

Lemma 6. *Let $\mathcal{L}$ be a collection of MSCs over $\mathcal{P}$. Then, there exists an XCMSG $G = (\Pi, M, h, \mathcal{A})$ which represents $\mathcal{L}$ iff there exists a regular set $L \subseteq \Sigma^*$ of semi well-formed words which is a representative semi-linearization of $\mathcal{L}$.*

Proof: Given an XCMSG G, we can fix some linearization of each atomic CMSC and realize a set of representative semi-linearizations as a homomorphic image of the language accepted by $\mathcal{A}$, where each letter is mapped to the fixed linearization of the CMSC corresponding to it. Since we finish all the events of a CMSC before moving to another CMSC, it is clear that the resulting words respect the local partial orders of the individual CMSCs. Hence this set is semi well-formed. The proof of the converse is similar to that of Lemma 1. □

Also, a result analogous to Lemma 4 goes through smoothly. A semi well-formed word $w \in \Sigma^*$ is said to be (b', b)-*bounded* if for every prefix x of w, the number of unmatched send-events in x of any type is at most b and the number

of unmatched receive-events in x of any type is at most b'. A language is said to be (b', b)-bounded if each word in it is (b', b)-bounded.

Lemma 7. *Given an MSO sentence φ and $b, b' \in \mathbb{N}$, the set of all semi well-formed words $w \in \Sigma^*$ such that w is (b', b)-bounded and $msc(w) \models \varphi$ is regular.*

We can also show that any regular language of semi well-formed words is (b', b)-bounded for some $b, b' \in \mathbb{N}$. Then, we can use Lemma 7 to give an alternative proof of Theorem 2.

References

AEY00. R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. 22nd Intl. Conf. on Software Engg.*, pages 304–313, 2000.

AY99. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. CONCUR '99*, vol. 1664 of *LNCS*. Springer-Verlag, 1999.

GMP01. E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proc. TACAS'01*, vol. 2031 of *LNCS*, pages 496–511. Springer-Verlag, 2001.

HMKT99. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Towards a theory of regular MSC languages. *BRICS Report RS-99-52, Department of Computer Science, Aarhus University, Denmark*, 1999.

HMKT00a. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. ICALP '00*, vol. 1853 of *LNCS*. Springer-Verlag, 2000.

HMKT00b. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Regular collections of Message Sequence Charts. In *Proc. MFCS '00*, vol. 1893 of *LNCS*. Springer-Verlag, 2000.

ITU97. ITU-TS Recommendation Z.120. Message sequence chart (MSC). *ITU-TS*, 1997.

Mad01. P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. ICALP '01*, vol. 2076 of *LNCS*. Springer-Verlag, 2001.

MM01. P. Madhusudan, B. Meenakshi. Beyond Message Sequence Graphs. *Institute of Mathematical Sciences Technical Report*, IMSc./2001/09/51. Available at `http://www.imsc.ernet.in/~madhu` or `/~bmeena`.

MR00. B. Meenakshi and R. Ramanujam. Reasoning about message passing in finite state environments. In *Proc. ICALP '00*, vol. 1853 of *LNCS*. Springer-Verlag, 2000.

Pel00. D. Peled. Specification and verification of message sequence charts. In *Proc. IFIP FORTE/PSTV*, 2000.

RGG96. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC, Vol. 28*, 1996.

Tho90. W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

Tho97. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.

TW97. P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proc. 12th IEEE Conf. on Logic in Computer Science*. IEEE Computer Society, 1997.

Grouping Techniques for One Machine Scheduling Subject to Precedence Constraints*

Monaldo Mastrolilli

IDSIA, Galleria 2, 6928 Manno, Switzerland
monaldo@idsia.ch

Abstract. We consider the problem of scheduling n jobs on a single machine. Each job has a release date, when it becomes available for processing, and, after completing its processing, requires an additional delivery time. Feasible schedules are further restricted by job precedence constraints, and the objective is to minimize the time by which all jobs are delivered. In the notation of Graham et al. [2], this problem is noted $1|r_j, prec|L_{\max}$. We develop a polynomial time approximation scheme whose running time depends only linearly on the input size. This linear complexity bound gives a substantial improvement of the best previously known polynomial bound [4].

1 Introduction

We shall study the following problem. There is a set of n jobs J_j $(j = 1, ..., n)$. Each job J_j must be processed without interruption for $p_j \geq 0$ time units on a single machine, which can process at most one job at a time. Each job has a release date $r_j \geq 0$, when it first becomes available for processing and, after completing its processing on the machine, requires an additional delivery time $q_j \geq 0$; if s_j $(\geq r_j)$ denotes the time J_j starts processing, it has been delivered at time $L_j = s_j + p_j + q_j$. Delivery is a *non-bottleneck* activity, in that all jobs may be simultaneously delivered. Feasible schedules are further restricted by job precedence constraints given by the partial order $\prec$, where $J_j \prec J_k$ means that job J_k must be processed after job J_j. Our objective is to minimize, over all possible schedules, the *maximum delivery time*, i.e., $L_{\max} = \max_j L_j$. The problem as stated is strongly NP-hard even if there are no precedence constraints [8]. In the notation of Graham et al. [2], this problem is noted $1|r_j, prec|L_{\max}$, while the problem without precedence constraints is denoted $1|r_j|L_{\max}$.

Since these scheduling problems are known to be hard to solve optimally, the research focuses on giving polynomial-time approximation algorithms that produce a solution close to the optimal one. Ideally, one hopes to obtain a family of polynomial algorithms such that for any given $\varepsilon > 0$ the corresponding algorithm is guaranteed to produce a solution with a value within a factor of $(1 + \varepsilon)$

* Supported by the "Metaheuristics Network", grant HPRN-CT-1999-00106, and by Swiss National Science Foundation project 20-63733.00/1, "Resource Allocation and Scheduling in Flexible Manufacturing Systems".

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 268–279, 2001.
© Springer-Verlag Berlin Heidelberg 2001

of the optimum value; such a family is called a polynomial time approximation scheme (PTAS).

Hall & Shmoys [3,5] propose two PTASs for problem $1|r_j|L_{\max}$, the running time of which are $O((\frac{n}{\varepsilon})^{O(1/\varepsilon)})$ and $O(n \log n + n(1/\varepsilon)^{O(1/\varepsilon^2)})$. They also give a PTAS for problem $1|r_j, prec|L_{\max}$ [4], which consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS for $1|r_j, q_j|C_{\max}$, where Δ denotes an upper bound on the optimal value of any given instance whose data are assumed to be integral. This polynomial running time should be contrasted with the time complexity of their result for problem $1|r_j|L_{\max}$, where they were able to achieve a considerably better time. To some extent, this is not surprising, since precedence constraints add a substantial degree of difficulty, and one important area of research in scheduling theory has been to study under what conditions a precedence-constrained problem is computationally harder than its counterpart with independent jobs.

In this paper we present a new PTAS for $1|r_j, prec|L_{\max}$ that runs in $O(n + \ell + 1/\varepsilon^{O(1/\varepsilon)})$ time, where ℓ denotes the number of precedences. This linear complexity bound is a substantial improvement compared to the above mentioned results. Note that the time complexity of this PTAS is best possible with respect to the input size. Moreover, the existence of a PTAS whose running time is also polynomial in $1/\varepsilon$ for a strongly NP-hard problem would imply P=NP [1]. We also improve the previous results [3,5] for problem $1|r_j|L_{\max}$ with respect to both, n and ε. Finally, we remark that the multiplicative constant hidden in the $O(n+\ell)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

To achieve these improvements a better and deeper analysis of the precedence structure is needed. We show that the precedence graph can be simplified into a more primitive graph. This simplification depends on the desired precision ε of approximation; the closer ε is to zero, the closer the modified graph should resemble the original one. With this simplified precedence structure it is possible to partition jobs into a constant number of subsets of jobs having similar precedence relations. Then, jobs belonging to the same subset can be grouped together into single jobs. The resulting instance is modified to obtain a constant number of instances with different release dates and delivery times; the final step consists of executing the extended Jackson's rule on this constant number of instances; the best schedule generated is output.

2 Structuring the Input

As first step to the construction of a PTAS for problem $1|prec, r_j|L_{\max}$, we discuss several techniques which add structure to the input data. Here the main idea is to turn a difficult instance into a more primitive instance that is easier to tackle. This simplification depends on the desired precision of approximation.

For simplicity, we shall assume that $1/\varepsilon$ is integral and $0 < \varepsilon < 1$. We start providing some lower and upper bounds on the optimal value *opt*. We define $P = \sum_{j=1}^{n} p_j$, $p_{\max} = \max_j p_j$, $r_{\max} = \max_j r_j$, $q_{\max} = \max_j q_j$ and

$d = \max_{j=1,\ldots,n} \{r_j + p_j + q_j\}$. Let $LB = \max\{P, d\}$, we claim that $LB \leq opt \leq 3LB$. Indeed, since P and d are lower bounds on opt, LB is also a lower bound on opt. We show that $3LB$ is an upper bound on opt by exhibiting a schedule with value at most $3LB$. Starting from time $r_{\max}$ all jobs have been released and they can be scheduled one after the other in any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. Then every job is completed by time $r_{\max} + P$ and the maximum delivery time is bounded by $r_{\max} + P + q_{\max} \leq P + 2d \leq 3LB$. By dividing every release, delivery and processing time by LB, we may (and will) assume, without loss of generality, that $LB = 1$ and

$$1 \leq opt \leq 3. \tag{1}$$

Following Lageweg, Lenstra & Rinnoy Kan [7], if $J_j \prec J_k$ and $r_j > r_k$, then we can reset $r_k := r_j$ and each feasible schedule will remain feasible. Similarly, if $q_j < q_k$ then we can reset $q_j := q_k$ without changing the objective function value of any feasible schedule. Thus, by repeatedly applying these updates we can always obtain an equivalent instance that satisfies

$$J_j \prec J_k \Longrightarrow (r_j \leq r_k \text{ and } q_j \geq q_k) \tag{2}$$

Such a resetting requires $O(\ell)$ time. In the following we will assume, w.l.o.g., that condition (2) holds.

A technique used by Hall and Shmoys [3] allows us to deal with only a constant number of release dates and delivery times. The idea is to round each release and delivery time down to the nearest multiple of $i\varepsilon$, for $i \in \mathbb{N}$. Since $r_{\max} \leq d \leq 1$, the number of different release dates and delivery times is now bounded by $1/\varepsilon + 1 < 2/\varepsilon$ (we are assuming $0 < \varepsilon < 1$). Clearly, the optimal value of this transformed instance cannot be greater than opt. Every feasible solution for the modified instance can be transformed into a feasible solution for the original instance just by adding ε to each job's starting time, and reintroducing the original delivery times. It is easy to see that the solution value may increase by at most 2ε. Thus in the remainder of this paper, without loss of generality, we shall restrict our attention to the case where there are only a constant number of release dates and delivery times.

Therefore, we will assume henceforth that the input instance has a constant number of release dates and delivery times, and that condition (2) holds. We shall refer to this instance as I. By the previous arguments, $opt \geq opt(I)$, where $opt(I)$ denotes the optimal value for instance I.

2.1 Partitioning Jobs

Partition the set of jobs in two subsets $L = \{J_j : p_j > \varepsilon\}$ and $S = \{J_j : p_j \leq \varepsilon\}$. Let us say that L is the set of *large* jobs, while S the set of *small* jobs. Observe that the number of large jobs is bounded by $P/\varepsilon \leq 1/\varepsilon$. We further partition the set S of small jobs as follows. For each small job $J_j \in S$ consider the following

three subsets of L:

$$Pre(j) = \{J_i \in L : J_i \prec J_j\},$$
$$Suc(j) = \{J_i \in L : J_j \prec J_i\},$$
$$Free(j) = L - (Pre(j) \cup Suc(j)).$$

Let us say that $T(j) = \{Pre(j), Suc(j), Free(j)\}$ represents a *3-partition of set* L with respect to job J_j. The number τ of distinct 3-partitions of L is clearly bounded by the number of jobs and by $3^{|L|} \leq 3^{1/\varepsilon}$, therefore $\tau \leq \min\left\{n, 3^{1/\varepsilon}\right\}$. Let $\{T_1, ..., T_\tau\}$ denote the set of all distinct 3-partitions. Now, we define the *execution profile* of a small job J_j to be a 3-tuple $< i_1, i_2, i_3 >$ such that $r_j = \varepsilon \cdot i_1$, $q_j = \varepsilon \cdot i_2$ and $T(j) = T_{i_3}$, where $i_1, i_2 = 0, 1, ..., 2/\varepsilon$ and $i_3 = 1, ..., \tau$.

Corollary 1. *The number π of distinct execution profiles is bounded by $\pi \leq \tau \cdot 4/\varepsilon^2 = 2^{O(1/\varepsilon)}$.*

Partition the set S of small jobs into π subsets, $S_1, S_2, ..., S_\pi$, such that jobs belonging to the same subset have the same execution profile. Clearly, $S = S_1 \cup S_2, ... \cup S_\pi$ and $S_h \cap S_i = \emptyset$, for $i \neq h$.

2.2 Adding New Precedences

Let us say that job J_h is a *neighbor* of set S_i $(i = 1, ..., \pi)$ if:

- J_h is a small job;
- $J_h \notin S_i$;
- there exists a precedence relation between job J_h and some job in S_i.

Moreover, we say that J_h is a *front-neighbor* (*back-neighbor*) of S_i if J_h is a neighbor of S_i and there is a job $J_j \in S_i$ such that $J_j \prec J_h$ ($J_h \prec J_j$).

Let $n_i = |S_i|$ $(i = 1, ..., \pi)$, and let $(J_{1,i}, ..., J_{n_i,i})$ denote any fixed ordering of the jobs from S_i that is consistent with the precedence relation. In the rest of this paper we consider the restricted problem in which the jobs from S_i are processed according to this fixed ordering. Furthermore, every back-neighbor (front-neighbor) J_h of S_i $(i = 1, ..., \pi)$ must be processed before (after) every job from S_i. This can be accomplished by adding a directed arc from $J_{j,i}$ to $J_{j+1,i}$, for $j = 1, ..., n_i - 1$, and by adding a directed arc from J_h to $J_{1,i}$, if J_h is a back-neighbor of S_i, or else an arc from $J_{n_i,i}$ to J_h, if J_h is a front-neighbor. Note that the number of added arcs can be bounded by $n + \ell$.

We observe that condition (2) is valid also after these changes. Indeed, if J_h is a back-neighbor of S_i then there is a job $J_j \in S_i$ such that $J_h \prec J_j$, and therefore by condition (2) we have $r_h \leq r_j$ and $q_h \geq q_j$. But, the jobs from S_i have the same release dates and delivery times, therefore $r_h \leq r_j$ and $q_h \geq q_j$ for each $J_j \in S_i$. It follows that if we restrict J_h to be processed before the jobs from S_i, condition (2) is still valid. Similar arguments hold if J_h is a front-neighbor. Moreover, all the jobs from S_i have the same release dates and delivery times, therefore condition (2) is still satisfied, if we restrict these jobs to be processed in any fixed ordering that is consistent with the precedence relation.

2.3 Grouping Small Jobs

Consider the jobs $(J_{1,i}, ..., J_{n_i,i})$ from subset S_i, for $i = 1, ..., \pi$, sorted according to the precedence relations. Let $J_{j,i}$ and $J_{j+1,i}$ be two consecutive jobs from S_i such that $p_{j,i} + p_{j+1,i} \leq \varepsilon$, for $j = 1, ..., n_i - 1$. We "group" together these two jobs to form a *grouped* job having the same release and delivery time as $J_{j,i}$ (and $J_{j+1,i}$), but processing time equal to the sum of the processing times of $J_{j,i}$ and $J_{j+1,i}$, i.e., $p_{j,i} + p_{j+1,i}$. (This is equivalent to say that we shall consider only those schedules where jobs $J_{j,i}$ and $J_{j+1,i}$ are processed together, i.e., $J_{j+1,i}$ after $J_{j,i}$.) Furthermore, the new grouped job must be processed after the predecessors of $J_{j,i}$ and before the successors of $J_{j+1,i}$. (Observe that, $J_{j,i}$ ($J_{j+1,i}$) excepted, the set of predecessors (successors) of $J_{j+1,i}$ ($J_{j,i}$) is the same as of $J_{j,i}$ ($J_{j+1,i}$).) We repeat this process, by using the modified set of jobs, until there are no more couples of consecutive jobs with sum of processing times less or equal to ε. The same procedure is performed for all other subsets S_i.

Let I_1 denote this new modified instance. Let us denote the corresponding processing times, release dates and delivery times of jobs in I_1 by p_j^1, r_j^1 and q_j^1, respectively. Observe that in instance I_1 the set of large jobs is the same as in I, and all the new grouped jobs are still "small", i.e., their processing times are not greater than ε. For simplicity of notation, let us use again L and S to denote, respectively, the set of large and small jobs with respect to instance I_1.

Lemma 1. *The number ν of jobs in instance I_1 is bounded by $\nu \leq \frac{1}{\varepsilon} + \frac{2}{\varepsilon} + 2\pi = 2^{O(1/\varepsilon)}$.*

Proof. Let us use S_i^1 to denote set S_i after the described grouping procedure, for $i = 1, ..., \pi$. Let $n_i^1 = |S_i^1|$. Observe that the sum $P_i^1 = \sum_{J_j \in S_i^1} p_j^1$ of processing times of the jobs from S_i^1 is equal to $P_i = \sum_{J_j \in S_i} p_j$.

First observe that every large job has processing time larger than ε and therefore there are at most $P/\varepsilon \leq 1/\varepsilon$ large jobs in I_1. By the way small jobs are grouped together, at the end of the described grouping procedure, the sum of processing times of any couple of consecutive jobs from S_i^1 cannot be smaller than ε. Therefore $\sum_{J_j \in S_i^1} p_j^1 \geq \left\lfloor \frac{n_i^1}{2} \right\rfloor \cdot \varepsilon$. Since $(\frac{n_i^1}{2} - 1) \cdot \varepsilon < \left\lfloor \frac{n_i^1}{2} \right\rfloor \cdot \varepsilon$, we have $\sum_{i=1}^{\pi} n_i^1 < 2(\pi + \sum P_i^1/\varepsilon) \leq 2(\pi + 1/\varepsilon)$. □

Note that instance I_1 can be computed in $O(n + \ell + 2^{O(1/\varepsilon)})$ time. Indeed, the time required to partition the set of jobs into π subsets can be bounded by $O(n + \ell + \pi)$; $O(n + \ell)$ is the time to add new precedences and group jobs.

We have already observed that the new precedence constraints added in Section 2.2 do not invalidate condition (2). Furthermore, let G denote a set of jobs that have been grouped together as described in this section. Then, all jobs from G have the same release date r_G and delivery time q_G, and set G has been replaced with a single job J_G having r_G and q_G as release date and delivery time, respectively. If there exist two jobs J_j and J_k such that $J_j \notin G$, $J_k \in G$ and $J_j \prec J_k$ ($J_k \prec J_j$), by condition (2), we know that $r_j \leq r_k$ and $q_j \geq q_k$ ($r_k \leq r_j$ and $q_k \geq q_j$), but, by construction, we have also $r_j \leq r_G$, $q_j \geq q_G$ and $J_j \prec J_G$

($r_G \leq r_j$, $q_G \geq q_j$ and $J_G \prec J_j$). Therefore, grouping jobs does not invalidate condition (2).

Lemma 2. *For each couple of jobs, J_j and J_k, of instance I_1 the following condition holds: $J_j \prec J_k \Longrightarrow (r_j^1 \leq r_k^1$ and $q_j^1 \geq q_k^1)$.*

3 The Main Algorithm

We make use of the following algorithm that is often called the *extended Jackson's rule* (since this generalizes a procedure to solve $1||L_{\max}$ due to Jackson [6]): schedule the jobs starting at the smallest r_j-value; at each decision point t given by a release date or a finishing time of some job, schedule a job j with the following properties: $r_j \leq t$, all its predecessors are scheduled, and it has the largest delivery time.

Our PTAS consists of executing the extended Jackson's rule on a constant number of instances obtained from I_1 by changing the release dates and delivery times of jobs; the best schedule generated is output. Without loss of generality, let us renumber jobs such that $p_1^1 \geq p_2^1 \geq ... \geq p_\lambda^1 \geq ... \geq p_\nu^1$, where λ denotes the number of large jobs. For $j = 1, .., \lambda$, let $R_j = \{r_j^1 + i\varepsilon : i \in \mathbb{N}$ and $r_j^1 + i\varepsilon \leq 3 - \varepsilon\}$. The release dates of large jobs $J_1, ..., J_\lambda$ are reset to new values taken from $R_1, ..., R_\lambda$, respectively. Depending on these values, the other release dates and delivery times may also change.

More precisely, our main algorithm performs the following steps.

(S-1) Initialize the solution *BestFound* to be the empty solution and set the corresponding value V to infinity.

(S-2) For each $(\rho_1, ..., \rho_\lambda) \in R_1 \times ... \times R_\lambda$ such that $\rho_j \geq \max_{J_h \in Pre(j)} \rho_h$ (for $j = 1, ..., \lambda$):

(S-2.1) Modify instance I_1 to get instance I_2 with release dates and delivery times equal to the following values

$$\begin{array}{ll} r_j^2 := \rho_j & \text{for } J_j \in L, \\ r_j^2 := \max\{r_j^1, \max_{h:J_h \in Pre(j)} r_h^2\}, & \text{for } J_j \in S, \\ q_j^2 := \max\{q_j^1, \max_{h:r_j^2 < r_h^2} q_h^2\} & \text{for } J_j \in L, \\ q_j^2 := \max\{q_j^1, \max_{h:J_h \in Suc(j)} q_h^2\}, & \text{for } J_j \in S. \end{array} \tag{3}$$

(S-2.2) Apply the extended Jackson's rule to instance I_2. Let Σ and $m(I_2, \Sigma)$ denote the solution and the solution value returned. If $m(I_2, \Sigma) < V$, then let $BestFound := \Sigma$ and $V := m(I_2, \Sigma)$.

(S-3) Return solution *BestFound* of value V.

Step (S-2.1) can be implemented as follows. Release dates and delivery times are updated separately. Consider any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. To update release dates, the jobs are processed in this order; when job J_j is processed, if J_j is a small job then r_j^2 is set to

$\max\{r_j^1, \max_{h:J_h \in Pre(j)} r_h^2\}$, otherwise to ρ_j. Let $J_1, ..., J_\nu$ denote any fixed ordering of the jobs that is consistent with the precedence relation and such that $r_1^2 \leq ... \leq r_\nu^2$. (Observe that in the following section we show that if $J_j \prec J_k$ then $r_j^2 \leq r_k^2$ (see Lemma 3). Therefore it is always possible to find an ordering of the jobs that is consistent with the precedence relation and such that $r_1^2 \leq ... \leq r_\nu^2$.) To update the delivery times, the jobs are processed in this reverse order, i.e., from ν to 1; when job J_j is processed, if J_j is small then q_j^2 is set to $\max\{q_j^1, \max_{h:J_h \in Suc(j)} q_h^2\}$, otherwise to $\max\{q_j^1, \max_{h:r_j^2 < r_h^2} q_h^2\}$.

4 Analysis

It is easy to see that $|R_j| \leq 3/\varepsilon$ for every $J_j \in L$. Therefore the number of different $(\rho_1, ..., \rho_\lambda)$ can be bounded by $(3/\varepsilon)^{1/\varepsilon}$, since the number λ of large job is not greater than $1/\varepsilon$. Recall that ν is the number of jobs in instance I_1, and therefore the number of precedences in I_1 is at most ν^2. Then, Steps (S-2.1) and (S-2.2) can be implemented to run in $O(\nu^2)$ time. Hence, the total running time of Steps (S-1)-(S-3) is $1/\varepsilon^{O(1/\varepsilon)}$.

The next lemma shows that condition (2) is valid for each instance I_2 considered in Step (S-2.1).

Lemma 3. *In Step (S-2.1), for each couple of jobs, J_j and J_k, of instance I_2 the following condition holds: $J_j \prec J_k \Longrightarrow (r_j^2 \leq r_k^2$ and $q_j^2 \geq q_k^2)$.*

Proof. We first prove by induction that

$$J_j \prec J_k \Longrightarrow r_j^2 \leq r_k^2. \tag{4}$$

Consider any fixed ordering $J_1, ..., J_\nu$ of the jobs that is consistent with the precedence relation. Trivially, condition (4) holds for set $\{J_1\}$. Assume that condition (4) holds for set $N_{k-1} = \{J_1, ..., J_{k-1}\}$, then we prove that condition (4) holds for set $N_k = \{J_1, ..., J_k\}$, for $2 \leq k \leq \nu$. If there is no job from set N_{k-1} that must be processed before J_k, then condition (4) holds for set N_k. Otherwise, we distinguish between the following cases:

1. If $J_k \in L$
   (a) and there is a large job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r_j^2 \leq r_k^2$ since $\rho_j \leq \rho_k$ (see Step (S-2));
   (b) and there is a small job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r_j^1 \leq r_k^1 \leq \rho_k = r_k^2$ by definition of R_k and by Lemma 2; furthermore $Pre(j) \subseteq Pre(k)$, since $J_j \prec J_k$, and $r_k^2 = \rho_k \geq \max_{J_h \in Pre(k)} \rho_h \geq \max_{J_h \in Pre(j)} \rho_h = \max_{J_h \in Pre(j)} r_h^2$; therefore

$$r_k^2 \geq \max\{r_j^1, \max_{h:J_h \in Pre(j)} r_h^2\} = r_j^2.$$

2. If $J_k \in S$

(a) and there is a large job J_j from set N_{k-1} such that $J_j \prec J_k$, then $J_j \in Pre(k)$ and therefore $r_j^2 \leq r_k^2$;
(b) and there is a small job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r_j^1 \leq r_k^1$ by Lemma 2; furthermore $Pre(j) \subseteq Pre(k)$ since $J_j \prec J_k$, and $\max_{J_h \in Pre(k)} r_h^2 \geq \max_{J_h \in Pre(j)} r_h^2$; therefore

$$r_k^2 = \max\{r_k^1, \max_{h:J_h \in Pre(k)} r_h^2\} \geq \max\{r_j^1, \max_{h:J_h \in Pre(j)} r_h^2\} = r_j^2.$$

Hence, we have proved that if $J_j \prec J_k$ then $r_j^2 \leq r_k^2$. This result guarantees that it is always possible to find an ordering of the jobs that is consistent with the precedence relation and such that $r_1^2 \leq ... \leq r_\nu^2$. Let $J_1, ..., J_\nu$ denote this ordering. In the following, we prove by induction that

$$J_j \prec J_k \Longrightarrow q_j^2 \geq q_k^2. \tag{5}$$

Trivially, condition (5) holds for set $\{J_\nu\}$. Assume that condition (5) is true for set $N_{j+1} = \{J_{j+1}, ..., J_\nu\}$, then we prove that condition (5) holds for set $N_j = \{J_j, ..., J_\nu\}$, for $1 \leq j \leq \nu - 1$. If there is no job from set N_{j+1} that must be processed after J_j, then condition (4) holds for set N_j. Otherwise, we distinguish between the following cases:

1. If $J_j \in L$
   (a) and there is a large job J_k from set N_{j+1} such that $J_j \prec J_k$, then $r_j^2 \leq r_k^2$, $q_j^1 \geq q_k^1$ by Lemma 2, and $\max_{h:r_j^2 < r_h^2} q_h^2 \geq \max_{h:r_k^2 < r_h^2} q_h^2$; it follows that $q_j^2 = \max\{q_j^1, \max_{h:r_j^2 < r_h^2} q_h^2\} \geq \max\{q_k^1, \max_{h:r_k^2 < r_h^2} q_h^2\} = q_k^2$;
   (b) and there is a small job J_k from set N_{j+1} such that $J_j \prec J_k$, then $Suc(k) \subseteq Suc(j)$, and since $J_1, ..., J_\nu$ denote an ordering of the jobs that is consistent with the precedence relation, we have $Suc(k) \subseteq Suc(j) \subseteq N_{j+1}$. In the previous case (1.a) we have shown that $q_j^2 \geq \max_{h:J_h \in Suc(j)} q_h^2$ and hence $q_j^2 \geq \max_{h:J_h \in Suc(k)} q_h^2$. By observing that $q_j^2 \geq q_j^1$ and $q_j^1 \geq q_k^1$ by Lemma 2, we have $q_j^2 \geq \max\{q_k^1, \max_{h:J_h \in Suc(k)} q_h^2\} = q_k^2$.
2. If $J_j \in S$
   (a) and there is a large job J_k from set N_{j+1} such that $J_j \prec J_k$, then $q_j^2 \geq q_k^2$ since $J_k \in Suc(j)$;
   (b) and there is a small job J_k from set N_{j+1} such that $J_j \prec J_k$, then $q_j^1 \geq q_k^1$ by Lemma 2; furthermore $Suc(j) \supseteq Suc(k)$ since $J_j \prec J_k$, and $\max_{J_h \in Suc(j)} q_h^2 \geq \max_{J_h \in Suc(k)} q_h^2$; therefore

$$q_j^2 = \max\{q_j^1, \max_{J_h \in Suc(j)} q_h^2\} \geq \max\{q_k^1, \max_{J_h \in Suc(k)} q_h^2\} = q_k^2.$$

□

Now, we examine an artificial situation that we shall use as tool in analyzing our algorithm. Let us focus on instance I and a particular optimal solution Σ^* for I, consisting of job starting times $s_1^*, s_2^*, ..., s_n^*$. If $i\varepsilon \leq s_j^* < (i+1)\varepsilon$, for

some $i \in \mathbb{N}$, then let $\rho_j^* := i\varepsilon$. Consider the modified instance I^* in which the processing times of all jobs J_j remain unchanged, while release r_j^* and delivery q_j^* times are set as follows

$$\begin{array}{ll} r_j^* := \rho_j^* & \text{for } J_j \in L, \\ r_j^* := \max\{r_j, \max_{h:J_h \in Pre(j)} r_h^*\}, & \text{for } J_j \in S, \\ q_j^* := \max\{q_j, \max_{h:r_j^* < r_h^*} q_h^*\} & \text{for } J_j \in L, \\ q_j^* := \max\{q_j, \max_{h:J_h \in Suc(j)} q_h^*\}, & \text{for } J_j \in S. \end{array} \tag{6}$$

By using the same arguments as in the proof of Lemma 3, we can easily show that condition (2) holds also for instance I^*. Furthermore, let $opt(I^*)$ denote the optimum value for I^*. Then we have the following

Lemma 4. *$opt(I^*) = opt(I)$.*

Proof. Instance I^* is obtained from I by increasing (or leaving unchanged) release dates and delivery times. Therefore, any feasible solution for I^* is also a feasible solution for I and $opt(I^*) \geq opt(I)$. The claim follows by proving that there exists a solution for I^* of value $\leq opt(I)$.

Consider the optimal solution Σ^* for I, consisting of job starting times $s_1^*, s_2^*, ..., s_n^*$. It is easy to check that $r_j^* \leq s_j^*$ $(j = 1, ..., n)$ and, therefore, we can schedule the jobs of instance I^* as in Σ^*: the starting time of job J_j is s_j^*, for $j = 1, ..., n$. Let J_c be the job that finishes last, i.e., its delivery is completed last, then the value of this solution is equal to $s_c^* + p_c + q_c^*$. If we prove that $s_c^* + p_c + q_c^* \leq opt(I)$, then the claim follows.

We prove that $s_c^* + p_c + q_c^* \leq opt(I)$ by induction. Let $J_1, ..., J_n$ denote any fixed ordering of the jobs that is consistent with the precedence relation and such that $r_1^* \leq ... \leq r_n^*$. (Note that this is possible since condition (2) holds for instance I^*.) If $c = n$, then $q_c^* = q_c$ and $s_c^* + p_c + q_c^* \leq opt(I)$. Otherwise, assume that $s_j^* + p_j + q_j^* \leq opt(I)$ for every $j = c+1, ..., n$ (induction hypothesis).

If J_c is a large job, let J_h denote the job with $r_c^* < r_h^*$ and $q_h^* = \max_{j:r_c^* < r_j^*} q_j^*$ (ties are broken arbitrarily). Since $r_c^* < r_h^*$ and $r_c^* = \rho_c^*$, it follows that $r_c^* \leq s_c^* < r_h^* \leq s_h^*$, and job c is processed before job h in Σ^*, i.e. $s_h^* \geq s_c^* + p_c$. From induction hypothesis we know that $s_h^* + p_h + q_h^* \leq opt(I)$, and hence $s_c^* + p_c + p_h + q_h^* \leq opt(I)$. Observe that $s_c^* + p_c + q_c \leq opt(I)$. It follows that $s_c^* + p_c + \max\{q_h^*, q_c\} \leq s_c^* + p_c + \max\{(p_h + q_h^*), q_c\} \leq opt(I)$. Note that $q_c^* = \max\{q_h^*, q_c\}$, and we have $s_c^* + p_c + q_c^* \leq opt(I)$.

Otherwise, if J_c is a small job, let J_h denote the job such that $J_h \in Suc(c)$ and $q_h^* = \max_{j:J_j \in Suc(c)} q_j^*$ (ties are broken arbitrarily). Since $J_h \in Suc(c)$, it follows that job J_c must be processed before job J_h, i.e. $s_h^* \geq s_c^* + p_c$. From induction hypothesis we know that $s_h^* + p_h + q_h^* \leq opt(I)$, and hence $s_c^* + p_c + p_h + q_h^* \leq opt(I)$. It follows that $s_c^* + p_c + \max\{q_h^*, q_c\} \leq s_c^* + p_c + \max\{(p_h + q_h^*), q_c\} \leq opt(I)$, since $s_c^* + p_c + q_c \leq opt(I)$. The claim follows by observing that $q_c^* = \max\{q_h^*, q_c\}$. □

By inequality (1) and by definition of large jobs, in any optimal solution the starting time of each large job cannot be later than $3 - \varepsilon$. Therefore, $\rho_j^* \in R_j$, for each $J_j \in L$, and in one of the iterations of step (S-2), we have $(\rho_1, ..., \rho_\lambda) =$

$(\rho_1^*, ..., \rho_\lambda^*)$. Now, for simplicity of notation, let us use I_2 to denote the modified instance of step (S-2.1) when $(\rho_1, ..., \rho_\lambda) = (\rho_1^*, ..., \rho_\lambda^*)$. Consider instance I_2 and ungroup all the jobs that have been grouped together in Section 2.3: if job J_j is part of a grouped job J_g then, when we ungroup, we assume that the release date r_j^2 and delivery time q_j^2 of job J_j are equal to r_g^2 and q_g^2, respectively. Let I_2^u denote the resulting instance.

Lemma 5. *In instance I_2^u we have $r_j^2 = r_j^*$ and $q_j^2 = q_j^*$, for $j = 1, ..., n$.*

Proof. In Section 2.2 we modified the precedence structure by adding new precedences among jobs. We claim that the introduction of these precedences do not change sets $Pre(.)$ and $Suc(.)$ of jobs. Indeed, recall that we restricted the problem such that every back-neighbor (front-neighbor) J_h of S_i $(i = 1, ..., \pi)$ must be processed before (after) every job from S_i. If J_h is a back-neighbor of S_i, then there exists a job $J_j \in S_i$ such that $J_h \prec J_j$. It follows that $Pre(h) \subseteq Pre(j)$ and $Suc(j) \subseteq Suc(h)$. Recall that the jobs from S_i have, by definition, the same sets $Pre(j)$ and $Suc(j)$. Therefore, by assuming $J_h \prec J_j$ for each $J_j \in S_i$, we do not change the set $Pre(j)$ of any job $J_j \in S_i$, neither we change the set $Suc(h)$ of J_h. Similarly, if J_h is a front-neighbor of S_i, then by assuming that J_h is processed after all jobs from S_i, we do not change the set $Suc(j)$ of any job $J_j \in S_i$, neither set $Pre(h)$. Furthermore, jobs from S_i were further restricted to be processed in any fixed ordering that is consistent with the precedence relation. Clearly, adding precedences among jobs J_j having the same sets $Pre(j)$ and $Suc(j)$, does not change these sets for J_j. Therefore, the precedences added in Section 2.2 do not change the execution profile of any job.

According to (6), it is easy to check that small jobs having the same execution profile are set to the same r_j^* and q_j^* values. Now, consider the following situation. Replace any subset H of jobs having the same execution profile, with a job J_H having the same execution profile as the jobs in H, and whose processing time is not greater that ε, i.e., J_H is a small job in the modified set of jobs. According to (6), update the release dates and delivery times of this modified set of jobs; then it is easy to see that $r_H^* = r_j^*$ and $q_H^* = q_j^*$ for each $J_j \in H$. The claim follows by observing that instance I_1 is obtained from I by adding new precedences which do not change the execution profile of any job, and by replacing subsets of jobs with new small jobs having the same execution profiles. □

Consider step (S-2.2), and let $m(I_2, \Sigma)$ denote the value of the solution Σ returned by the extended Jackson's rule when applied to instance I_2. In order to show that the described algorithm is a PTAS, it is sufficient to prove the following

Lemma 6. $m(I_2, \Sigma) \leq (1 + \varepsilon)opt$.

Proof. We start by making two simple observations. First, consider any set C of jobs belonging to instance I_2. Then ungroup the jobs from C and let C^u denote the resulting set of jobs. By Lemma 5, we have the following equation

$$\min_{J_j \in C} r_j^2 + \sum_{J_j \in C} p_j^2 + \min_{J_j \in C} r_j^2 = \min_{J_j \in C^u} r_j^* + \sum_{J_j \in C^u} p_j + \min_{J_j \in C^u} q_j^*. \tag{7}$$

Second, we observe that

$$opt(I^*) \geq \min_{J_j \in C^u} r_j^* + \sum_{J_j \in C^u} p_j + \min_{J_j \in C^u} q_j^* \tag{8}$$

Let us define a critical job J_c as one that finishes last in Σ, i.e., its delivery is completed last. Associated with a critical job J_c there is a *critical sequence* B consisting of those jobs tracing backward from J_c to the first idle time in the schedule. Let us fix a critical job J_c and denote the last job J_b in the critical sequence with $q_c^2 > q_b^2$ as *interference job* for the critical sequence. Let C denote the set of jobs processed after J_b in the critical sequence. By the way that J_b was chosen, clearly $q_k^2 \geq q_c^2$ for all jobs $J_k \in C$.

We claim that if there is no interference job then $m(I_2, \Sigma) \leq opt$, otherwise $m(I_2, \Sigma) \leq opt + p_b^2$. Assume that an interference job J_b exists. Then, let s_b denote the starting time of job J_b, then $m(I_2, \Sigma) = s_b + p_b^2 + \sum_{J_j \in C} p_j^2 + q_c^2$. Let us say that a job is *available* at time t if it has been released and all its predecessors are scheduled at time t. Notice that for the extended Jackson's rule to schedule job J_b at time s_b implies that no job $J_k \in C$ could have been available at time s_b, since otherwise such a job J_k would have taken priority over job J_b. Now, there are two alternative situations that could make a job $J_k \in C$ not available at time s_b: (1) $r_k^2 \leq s_b$ but there is a predecessor J_j of J_k that has not been yet scheduled at time s_b; (2) no job from C has been released at time s_b. Consider case (1) and, without loss of generality, let J_j denote some not yet scheduled predecessor of J_k for which all its predecessors are scheduled. By Lemma 3, we have $r_j^2 \leq r_k^2$ and $q_j^2 \geq q_k^2$. Recall that $q_k^2 \geq q_c^2$ for all $J_k \in C$, and $q_c^2 > q_b^2$ by definition of interference job. Therefore job J_j has $q_j^2 > q_b^2$, $r_j^2 \leq s_b$, and it is available at time s_b since all its predecessors are scheduled. By the definition of the extended Jackson's rule, no unscheduled job J_j could have been available at the time when J_b is processed with $q_j^2 > q_b^2$, since otherwise such a job J_j would have taken priority over job J_b. Therefore case (1) is not possible. According to case (2) we have $s_b < \min_{J_j \in C} r_j^2$. Furthermore, by the way that J_b was chosen, clearly $q_c^2 \leq \min_{J_j \in C} q_j^2$. Now we can bound the value of $m(I_2, \Sigma)$, using (7) and (8), as

$$\begin{aligned} m(I_2, \Sigma) &\leq p_b^2 + \min_{J_j \in C} r_j^2 + \sum_{J_j \in C} p_j^2 + \min_{J_j \in C} q_j^2 \\ &\leq p_b^2 + opt(I^*) \leq p_b^2 + opt. \end{aligned}$$

If there is no interference job, using (7) and (8), it is easy to check that $m(I_2, \Sigma) = \min_{J_j \in B} r_j^2 + \sum_{J_j \in B} p_j^2 + \min_{J_j \in B} q_j^2 \leq opt$.

Then if there is no interference job or J_b is a small job we have $m(I_2, \Sigma) \leq (1 + \varepsilon) \cdot opt$, by definition of small jobs (grouped or not). Now, consider the last case in which the interference job J_b is a large job. Recall that $q_c^2 > q_b^2$ and $s_b < \min_{J_j \in C} r_j^2$, thus $r_c^2 > r_b^2$. But J_b cannot be a large job since according to (3) if $r_c^2 > r_b^2$ then $q_b^2 \geq q_c^2$. □

By the arguments of Section 2, solution Σ can be easily transformed into a feasible solution for the original instance. This results in a $(1+3\varepsilon)$-approximate solution, and therefore we have given a polynomial time approximation scheme for $1|r_j, prec|L_{\max}$.

Theorem 1. *There exists a PTAS for problem* $1|r_j, prec|L_{\max}$ *that runs in* $O(n + \ell + 1/\varepsilon^{O(1/\varepsilon)})$ *time.*

References

1. M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness.* W.H. Freeman, 1979.
2. R. Graham, E. Lawler, J. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. volume 5, pages 287–326. North–Holland, 1979.
3. L. A. Hall and D. B. Shmoys. Approximation algorithms for constrained scheduling problems. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS 1989)*, pages 134–139, 1989.
4. L. A. Hall and D. B. Shmoys. Near-optimal sequencing with precedence constraints. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference (IPCO 1990)*, pages 249–260. University of Waterloo Press, 1990.
5. L. A. Hall and D. B. Shmoys. Jackson's rule for single-machine scheduling: Making a good heuristic better. *MOR: Mathematics of Operations Research*, 17:22–35, 1992.
6. J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report Research Report 43, Management Science Research Project, UCLA, 1955.
7. B. J. Lageweg, J. K. Lenstra, and A. H. G. R. Kan. Minimizing maximum lateness on one machine: Computational experience and some applications. *Statist. Neerlandica*, 30:25–41, 1976.
8. J. K. Lenstra, A. H. G. R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Operations Research*, 1:343–362, 1977.

Properties of Distributed Timed-Arc Petri Nets

Mogens Nielsen[1], Vladimiro Sassone[2,⋆], and Jiří Srba[1,⋆⋆]

[1] **BRICS**⋆⋆⋆, Dept. of Computer Science, University of Aarhus
[2] University of Sussex

Abstract. In [12] we started a research on a distributed-timed extension of Petri nets where time parameters are associated with tokens and arcs carry constraints that qualify the age of tokens required for enabling. This formalism enables to model e.g. hardware architectures like GALS. We give a formal definition of process semantics for our model and investigate several properties of local versus global timing: expressiveness, reachability and coverability.

Introduction

Verification of concurrent and parallel systems plays nowadays an important role in the concurrency theory, with a number of successful applications. Algorithmic methods have been developed for process algebras generating infinite state systems, timed process algebra, Petri nets, lossy vector addition systems, counter machines, real time systems and many others. In particular, the idea to equip automata with real time appeared to be very fruitful and there are even automatic verification tools for such systems as UPPAAL [9] and KRONOS [5].

The main idea behind timed automata is to equip a standard automaton with a number of synchronous clocks, and to allow transitions to be conditioned on clock values and to affect (reset) clocks. One of the objections to this formalism is the assumption of perfect synchrony between clocks. For many applications this assumption is justified, but for others it is unrealistic. Clearly, geographically highly distributed systems are prime examples, but the issue has been addressed also for hardware design; e.g. in work on so-called Globally Asynchronous Locally Synchronous (GALS) systems [11].

Following these arguments we suggested in [12] a new model called *distributed timed-arc Petri nets* (DTAPN). One of the reasons for choosing Petri net formalism is the explicit representation of locality.

Several models that take time features into account have been presented in the literature (for a survey see [4,17]). For example *timed transitions Petri nets* were proposed in [13] where transitions are annotated with their durations. A model in which time parameters are associated to places is the *timed places*

⋆ Author partly supported by MUST project *TOSCA*.
⋆⋆ Author partly supported by the GACR, grant No. 201/00/0400.
⋆⋆⋆ Basic Research in Computer Science, Centre of the Danish National Research Foundation.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 280–291, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Petri nets of [16]. We shall analyse *timed-arc Petri nets* [3,7], a time extension of Petri nets where time (age) is associated to tokens and transitions are labelled by time intervals which restrict the age of tokens that can be used to fire them. In this model, time is considered to be *global*, i.e., all tokens grow older with the same speed. In spite of the fact that reachability is decidable for ordinary Petri nets [10], it is undecidable for global timed-arc Petri nets [15]. On the other hand, coverability is decidable for such a model [14,1], which is also known to offer 'weak' expressiveness, in the sense that it cannot simulate Turing machines [2].

In [12] a new model is suggested where time elapses in a place independently on other places, taking the view that places represent 'localities'. The idea of local clocks is generalised in the DTAPN model, where we use an equivalence relation on places to specify which pairs of places must synchronise. As special instances we get *local* timed-arc Petri nets (LT nets), where no synchronisations are forced, and *global* timed-arc Petri nets (GT nets), with full synchronisation.

In this paper we give a formal definition of process semantics which provides a reading of the differences between the LT and the GT net models and of their relative strengths. Among the motivations behind LT nets, they seem to be a weaker model than the global time one and some interesting properties could be verified algorithmically. Nevertheless, we prove that the general reachability problem for LT nets is undecidable. However, we show that a small modification of the problem (a slight restriction of the set of allowed initial markings) makes reachability decidable for LT nets, but not for GT nets. Finally, we argue that coverability is decidable for all DTAPNs.

1 Distributed Timed-Arc Petri Nets

Definition 1 (Distributed timed-arc Petri net).
A *place/transition Petri net* (PT) is a tuple (P, T, F), where P is a finite set of *places*, T is a finite set of *transitions* such that $T \cap P = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*.

A *distributed timed-arc Petri net* (DTAPN) is a tuple $N = (P, T, F, c, E, D)$, where (P, T, F) is a Petri net and:

- $c : F|_{P \times T} \to D \times (D \cup \{\infty\})$ is a *time constraint* on transitions such that for each arc $(p, t) \in F$, if $c(p, t) = (t_1, t_2)$ then $t_1 \leq t_2$,
- $E \subseteq P \times P$ is an equivalence relation on places (*synchronisation relation*),
- $D \in \{\mathbb{R}_0^+, \mathbb{N}_0\}$ is either *continuous* or *discrete* time.

Let $x \in D$ and $c(p, t) = (t_1, t_2)$. We write $x \in c(p, t)$ whenever $t_1 \leq x \leq t_2$. We also define ${}^\bullet x = \{y \mid (y, x) \in F\}$, $x^\bullet = \{y \mid (x, y) \in F\}$, for $x \in P \cup T$ and use $\mathcal{B}(X)$ to denote the set of all finite multisets on a set X. In what follows, we assume that ${}^\bullet t \neq \emptyset$ for every $t \in T$.

A *marked* PT net is a net (P, T, F) together with an *initial marking* $M \in \mathcal{B}(P)$. A *marking* of a DTAPN (P, T, F, c, E, D) is a function $M \colon P \to \mathcal{B}(D)$. A

marked DTAPN is a pair (N, M), for M a marking of N with all tokens of age 0. Each place is thus assigned a number of tokens, and each token is annotated with a real (natural) number (*age*). Let $x \in \mathcal{B}(D)$ and $a \in D$. We define $x \leftarrow\!\!+\, a$ to add the value a to every element of x, i.e., $x \leftarrow\!\!+\, a = \{b + a \mid b \in x\}$.

The dynamics of DTAPNs is defined by two types of transition relations: *firing* of a transition and *time-elapsing*.

Definition 2 (Transition rules).
Let $N = (P, T, F, c, E, D)$ be a DTAPN, M a marking and $t \in T$.

- We say that t is *enabled* by M iff $\forall p \in {}^\bullet t.\ \exists x \in M(p).\ x \in c(p,t)$.
- If t is enabled by M, it can *fire* producing a marking M', in symbols $M[t\rangle M'$, such that:

$$\forall p \in P.\ M'(p) = \Big(M(p) \setminus C^-(p,t)\Big) \cup C^+(t,p)$$

  where C^- and C^+ are chosen to satisfy the following equations (which may have several solutions):

$$C^-(p,t) = \begin{cases} \{x\} \text{ such that } x \in M(p) \text{ and } x \in c(p,t) & \text{if } p \in {}^\bullet t \\ \emptyset & \text{otherwise} \end{cases}$$

$$C^+(t,p) = \begin{cases} \{0\} & \text{if } p \in t^\bullet \\ \emptyset & \text{otherwise.} \end{cases}$$

  Note that the new tokens added to places $t^\bullet$ are of initial age 0.
- We define a *time-elapsing* transition ϵ, for $\epsilon\colon P/E \to D$, as follows, where $[p]_E$ denotes the E-equivalence class of p:

$$M[\epsilon\rangle M' \quad \text{iff} \quad \forall p \in P.\ M'(p) = M(p) \leftarrow\!\!+\, \epsilon([p]_E).$$

We write $M \longrightarrow M'$ iff either $M[t\rangle M'$ or $M[\epsilon\rangle M'$ for some t or ϵ.

Two classes of DTAPNs play prominent roles. The first one requires an absolute synchronisation and was studied in the past, while the other one is a new model suggested in [12] — completely asynchronous:

- *Global timed-arc Petri nets (GT nets)*: $E = P \times P$,
- *Local timed-arc Petri nets (LT nets)*: $E = \Delta_P = \{(p,p) \mid p \in P\}$.

2 GALS Architectures

In high-performance VLSI the clock management is the main source of power consumption. Keeping one global clock synchronised is usually the bottleneck of a processor design. In [11] the authors suggest a method to decrease the pitfalls of global clock distribution. A processor design is partitioned into synchronous blocks that communicate globally with each other on asynchronous basis using a handshake mechanism. This architecture is called *Globally Asynchronous and*

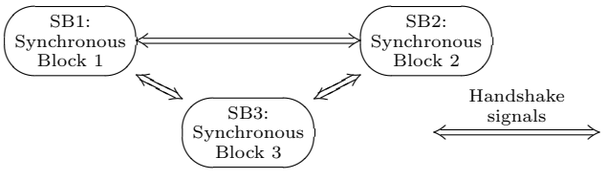


Fig. 1. GALS architecture

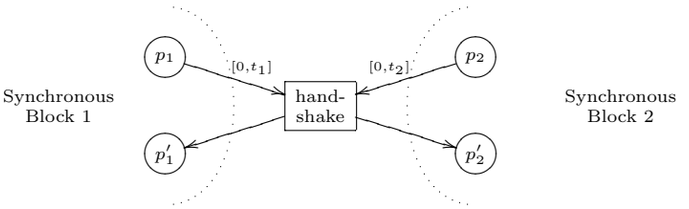


Fig. 2. Modelling of handshake mechanism between SB1 and SB2

Locally Synchronous (GALS) architecture. The authors applied this technology to a realistic design with million gates, saving about 30% of power energy with negligible overhead. Fully asynchronous solutions to the problem have been also examined, for an overview see e.g. [8].

Distributed timed-arc Petri nets, in particular LT nets in the fully asynchronous case, appear to be a good model for such architectures. Let us now focus on the design of GALS. Figure 1, from [11], represents the basic concept of GALS architecture for three synchronous components. For each of the synchronous blocks SB1, SB2 and SB3 we design a GT net, joining them together by means of a handshake communication as in Figure 2 (in which every arc from a place p to a transition t is labelled by the time interval $c(p,t)$). This creates the final DTAPN with a synchronisation relation respecting the place partitioning given by the blocks SB1, SB2 and SB3. The transition 'handshake' forces the blocks SB1 and SB2 to synchronise and after this transition is fired new tokens of age 0 appear in places p'_1 and p'_2. Then SB1 and SB2 can continue their un-synchronised performance. If we set $t_1 = t_2 = \infty$ then there are no time constrains on the maximal waiting time for the handshake communication. However, by changing the values t_1 and t_2 we may forbid a late handshake synchronisation.

Further examples of LT nets and DTAPNs, as e.g. timed producer/consumer systems or Fischer's mutual exclusion protocol, have been described in [12].

3 Process Semantics for DTAPN

We aim at providing a common ground on which to assess relative expressiveness of GT nets and LT nets. In this section, building upon the idea of PT net

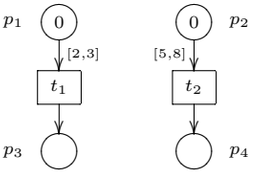


Fig. 3. Dependent transitions in a GT net and independent in an LT net

processes [6], we formalise a notion of processes of DTAPNs and establish their properties with respect to firing sequences.

The subtle differences between computations of LT and GT nets that we want to address can be illustrated with the help of the net of Figure 3. Were this net an ordinary net, transitions t_1 and t_2 would be completely independent. Things are not so neat when we consider the time constraints. If the net is a GT net, i.e. time is global, after firing t_2, the transition t_1 cannot possibly fire anymore. If instead we consider the net under the local time interpretation, t_1 and t_2 can again be considered completely independent, as the one's firing cannot affect the other's enabledness.

Definition 3 (Process Nets).
A *process net* is a PT net $\Pi = (P, T, F)$ such that Π is acyclic, i.e., for $x, y \in \Pi$, $x \prec_\Pi y$ implies $y \not\prec_\Pi x$, and Π is deterministic, i.e., for each $p \in P$, $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$, where $\prec_\Pi$ denotes the transitive closure of F.

Each place p of a process net Π has exactly zero or one transition in its preset. We define ${}^\star p = t$ if ${}^\bullet p = \{t\}$ and ${}^\star p = \bot$ if ${}^\bullet p = \emptyset$. By $\min(\Pi)$ we denote the set of $\prec_\Pi$-minimal places of Π, i.e., $\min(\Pi) = \{p \in P \mid {}^\star p = \bot\}$. Process nets are implicitly considered marked, with initial marking $\min(\Pi)$. With abuse of notation, in the following we shall write $p \in \Pi$ and $t \in \Pi$ avoiding explicit mention of the components P and T of Π, as this is not likely to create ambiguity. Analogously, we usually drop the subscript from $\prec_\Pi$.

Definition 4 (PT Process).
A *map* $\sigma\colon (P, T, F, M) \to (P', T', F', M')$ of marked PT Petri nets is a function $\sigma\colon P \cup T \to P' \cup T'$ mapping P to P' and T to T' such that $\sigma(M) = M'$, and for all $t \in T$, $\sigma({}^\bullet t) = {}^\bullet \sigma(t)$ and $\sigma(t^\bullet) = \sigma(t)^\bullet$.
A *process* π of (N, M) is a map $\pi\colon \Pi \to (N, M)$, for Π a finite process net.

The notion of slice, which provides a snapshot of a running process' state, plays a role in our development. We say that $x, y \in \Pi$ are *concurrent* if neither $x \prec y$ nor $y \prec x$. A *slice* of π is a maximal set of concurrent places of Π. E.g., the PT net underlying the net of Figure 3 has precisely four slices: $\{p_1, p_2\}$, $\{p_1, p_4\}$, $\{p_2, p_3\}$, $\{p_3, p_4\}$. We use $S_\prec = \{t \mid t \prec s, s \in S\}$ and $S^\prec = \{t \mid s \prec t, s \in S\}$ to indicate the two parts a slice S partitions the transitions of Π into.

Processes of DTAPNs rest upon the notion of PT net process, enriching it with a suitable treatment of time constraints. Each firing of a transition is time-stamped with the time elapsed since the process began, according to each of the 'clocks' (E-equivalence classes) involved.

Definition 5 (DTAPN Net Processes).
Let $N = (P, T, F, c, E, D, M)$ be a marked DTAPN. A process of N is a process $\pi\colon \Pi \to N$ of the underlying PT net together with a $\lesssim$-totally preordered family $\delta = \{\delta_t\colon P/E \rightharpoonup D\}_{t\in\Pi}$ of partial functions such that $\delta_t(x)$ is defined if and only if $\pi({}^\bullet t \cup t^\bullet) \cap x \neq \emptyset$ and for each arc (p, t) of Π

$$\delta_t([\pi(p)]_E) - \delta_{\star p}([\pi(p)]_E) \in c(\pi(p), \pi(t)),$$

where, by convention, $\delta_\perp(x) = 0$ for all x, and where $\delta_t \lesssim \delta_{t'}$ if $\delta_t(x) \leq \delta_{t'}(x)$ for all $x \in \mathrm{dom}(\delta_t) \cap \mathrm{dom}(\delta_{t'})$.

The condition above enforces the time constraints c on the arcs by bounding appropriately the difference between tokens' creation and consumption times. The special case of $\delta_\perp$ deals with 0-aged tokens in the initial marking. For GT nets, each δ_t reduces to a single time-stamp according to the (unique) global clock. In the case of multiple clocks, the preorder $\lesssim$ ensures that the time domain is consistent, ruling out situations in which concurrent transitions have incompatible perceptions of the time elapsed. Notice that the linearity condition does not mean sequential processes, as we may have both $\delta_t \lesssim \delta_{t'}$ and $\delta_{t'} \lesssim \delta_t$.

Slices need refinement to adapt to our timed model. Observe, in fact, that $\{p_1, p_4\}$ can never be a slice of any process when the net in Figure 3 is considered as a GT net, as the behaviour in which t_2 occurs before t_1 is not realisable. We shall thus define a slice of a DTAPN process to be a slice S of the underlying PT process such that $\delta_t \lesssim \delta_{t'}$, for all $t \in S_\prec$ and all $t' \in S^\prec$.

We now proceed to prove an important sanity condition for our processes, by relating them to firing sequences and markings. In order to extract a marking from a slice, the following definition determines the age of tokens in each places as the difference between the time-stamp of the slice according to clock x (viz. $\max(S, x)$) and the time (according to the same x) when the token was generated. This allows us to pass from the absolute time on processes' transitions to the relative one found in firing sequences' markings.

Definition 6 (Markings Compatible with a Slice).
Let $(\pi\colon \Pi \to N, \delta)$ be a process of a marked DTAPN N and S a slice thereof. Marking M_S is associated to S if only places in $\pi(S)$ are marked, and for each $\bar{p} \in \pi(S)$,

$$M_S(\bar{p}) = \{\max(S, [\bar{p}]_E) - \delta_x([\bar{p}]_E) \mid x = {}^\star p,\ \pi(p) = \bar{p}\}$$

where $\max(S, x) = \max\{\delta_t(x) \mid t \in {}^\bullet S,\ x \in \mathrm{dom}(\delta_t)\}$, convening that $\max \emptyset = 0$. The set of markings of N compatible with S is

$$m(\pi, S) = \{M \mid M_S[\epsilon\rangle M,\ \text{for } \epsilon \text{ a } \textit{time-elapsing} \text{ transition of } N\}.$$

Theorem 1. *Let (N, M) be a DTAPN and (π, δ) a process thereof. For each slice S of π and each $M' \in m(\pi, S)$ there exists a firing sequence $M \longrightarrow^* M'$.*

Proof. By induction on the size of $S_\prec$. The base case is easy, for $S_\prec = \emptyset$. In the induction step, we must have $t \in S_\prec$ with $t^\bullet \subseteq S$. Among these, choose t

one with $\lesssim$-maximal δ_t, that exists by hypothesis on δ, so to ensure that $S \setminus t^\bullet$ is a slice. By induction hypothesis, there is a firing sequence $M \to^* M_1$, where M_1 is a marking in $m(\pi, S \setminus t^\bullet)$ such that $M_1[t\rangle M_2$. Then, by an appropriate time-elapsing transition, $M_2[\epsilon\rangle M'$, as required. □

Theorem 2. *Let (N, M) be a DTAPN. For each firing sequence $M \longrightarrow^* M'$ of N, there exists a process (π, δ) such that $M' \in m(\pi, S)$, for S a slice of π.*

Proof. Easy, by induction on the length of the firing sequence. □

The difference between GT nets and LT nets is reflected in our formalisation above in two related aspects. Firstly, GT nets have fewer processes, due to the more stringent synchronisation constraints. Secondly, these processes have fewer slices, that is a smaller internal concurrency. This is nicely summarised in $\lesssim$, that is a 'loose' preorder in the case of LT nets, and essentially becomes a 'tight' linear order for GT nets.

4 Reachability and Coverability

LT and GT nets can be compared on the grounds of various decidability questions. Ruiz, Gomez and Escrig recently proved in [15] that reachability is undecidable for GT nets. Their proof does not imply undecidability for LT nets, because it relies on synchronised places. In principle, it may seem that the model of LT nets is less powerful than the one of GT nets. Nevertheless, we demonstrate that reachability for LT nets is undecidable as well. The proof is based on a reduction from the halting problem of Minsky machine with two counters. Notice that this contrasts with the result by Mayr [10] stating the decidability of reachability for ordinary Petri nets. The reachability problem for local timed-arc Petri nets can be formulated as follows.

Problem:	Reachability for LT nets.
Instance:	A marked LT net (N, M) and a final marking M'.
Question:	$M \longrightarrow^* M'$?

Definition 7 (Minsky machine with two counters).
Minsky machine R with two counters c_1 and c_2 is a finite sequence

$$R = (L_1 : I_1, \quad L_2 : I_2, \quad \ldots, \quad L_n : I_n)$$

where $L_1, \ldots, L_n$ are pairwise different *labels*, $I_1, \ldots, I_n$ are *instructions* such that $I_1, \ldots, I_{n-1}$ are exactly of one of the following types:

- *increment:* $c_r := c_r + 1$; `goto` L_j
- *test and decrement:* `if` $c_r = 0$ `then goto` L_j `else` $c_r := c_r - 1$; `goto` L_k

where $1 \leq r \leq 2$ and $1 \leq j, k \leq n$. The last instruction I_n is always a special instruction `halt`.

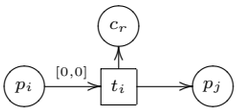


Fig. 4. Increment instruction

A machine R starts its execution (with given input values of c_1 and c_2) from the instruction I_1 and it *halts* if it reaches the instruction `halt` in a finite number of steps. Otherwise it *diverges*. The halting problem for a machine R with the initial values of counters $c_1 = c_2 = 0$ is known to be undecidable. The following variant of the problem is easily seen to be undecidable as well.

Problem:	Halting problem with empty counters.
Instance:	A Minsky machine R with $c_1 = c_2 = 0$.
Question:	Does R halt and both counters are empty?

Given a Minsky machine $R = (L_1 : I_1,\ \ L_2 : I_2,\ \ \ldots,\ \ L_n : I_n)$ we construct a local timed-arc Petri net N with continuous time which weakly simulates the machine R. We define $N = (P, T, F, c, \Delta_P, \mathbb{R}_0^+)$ where

$$\begin{aligned} P = \quad & \{p_i \mid 1 \le i < n,\ I_i \text{ of type increment}\} \\ & \cup \{p_i, p_i^1, p_i^2, p_i^3, p_i^4, p_i^5 \mid 1 \le i < n,\ I_i \text{ of type test and decrement}\} \\ & \cup \{p_n, c_1, c_2, p_s, p_e\}, \end{aligned}$$

$$\begin{aligned} T = \quad & \{t_i \mid 1 \le i < n,\ I_i \text{ of type increment}\} \\ & \cup \{t_i, t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6 \mid 1 \le i < n,\ I_i \text{ of type test and decrement}\} \\ & \cup \{t_s, t_e\}, \end{aligned}$$

F and c are described in the text below.

For every instruction $L_i : c_r := c_r + 1$; `goto` L_j, with $1 \le i < n$ and $1 \le r \le 2$, we add the arcs between places and transitions depicted in Figure 4. For every instruction L_i : `if` $c_r = 0$ `then goto` L_j `else` $c_r := c_r - 1$; `goto` L_k, with $1 \le i < n$ and $1 \le r \le 2$, we add the arcs depicted in Figure 5. Moreover we add a starting and an ending transition, as illustrated in Figure 6. Initial and final markings M and M' are

$$M(p) = \begin{cases} \{0\} & \text{if } p = p_s \\ \{0,0\} & \text{if } p \in \{c_1, c_2\} \\ \emptyset & \text{otherwise,} \end{cases} \qquad M'(p) = \begin{cases} \{0\} & \text{if } p = p_e \\ \emptyset & \text{otherwise.} \end{cases}$$

We prove that $M \longrightarrow^* M'$ if and only if R halts on the input $c_1 = c_2 = 0$ with both counters empty. Thus we show that reachability for LT nets is undecidable.

Lemma 1. *If R halts with both counters empty then $M \longrightarrow^* M'$.*

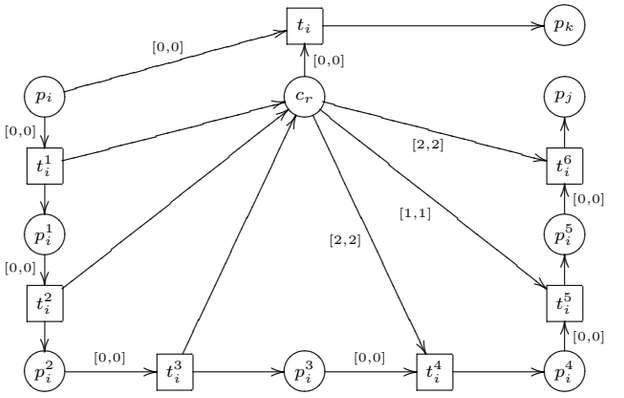


Fig. 5. Test and decrement instruction

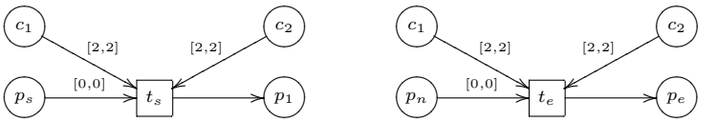


Fig. 6. Start and end of the simulation

Proof. Suppose that after a finite sequence of instructions executed by R the machine stops in the instruction L_n : `halt` with both counters empty. Then we can simulate this sequence in N as follows.

First, we let the four tokens in places c_1 and c_2 reach the age 2 and then we fire the starting transition t_s, which puts a token into place p_1 and in both c_1 and c_2 remains one token of age 2. An increment instruction I_i is simulated by firing the transition t_i without any time elapsed. If I_i is a test and decrement instruction and the corresponding place c_r contains a token of age 0, we fire the transition t_i. Again there is no time-elapsing transition. If the place c_r contains only one token of age 2 then we fire the sequence of transitions $t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6$. First, three tokens of age 0 are added and then the token of age 2 is removed by the transition t_i^4. Then we allow to pass one time unit in the place c_r, which means that c_r now contains three tokens of age 1. We consume one of them by firing the transition t_i^5 and let pass another time unit in c_r. Then we fire the transition t_i^6. The resulting marking contains one token of age 2 in c_r, one token of age 0 in p_j and the place c_{3-r} is untouched. Eventually a token of age 0 appears in the place p_n and the places c_1 and c_2 contain one token of age 2 each. That means that we can fire the ending transition t_e and reach M'. □

Lemma 2. *If $M \longrightarrow^* M'$ then the machine R halts with both counters empty.*

Proof. We can naturally simulate the behaviour of the net N by executing the corresponding instructions of the machine R. The only problematic case is when

a transition t_i^1 is fired and the counter c_r is non-empty. However, we show that if this happens then the marking M' cannot be reached.

First observe that the only transition that can be fired from M is t_s and there must be a time-elapsing transition before it. Thus the resulting marking contains one token of age 0 in p_1 and one token of age 2 in both c_1 and c_2. Notice that whenever a token of age strictly greater than 2 appears in c_1 or c_2 (we call such a token *dead*), M' is not reachable. The same happens if a token of age different from 0 appears in some of the places $p_1, p_2, \ldots, p_n$. Thus a time-elapsing transition cannot occur if we aim to reach the marking M'. The values of counters c_1 and c_2 are represented by the corresponding number of tokens of age 0 in the places c_1 and c_2 respectively, with one additional token of age 2. Suppose that we fire a 'cheating' sequence $t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6$ such that c_r contains except for one token of age 2 also a non-zero number of tokens of age 0. By examining all possibilities of firing this sequence (we want to avoid dead tokens), we end up with having at least two tokens of age 2 in c_r and moreover all tokens in c_r are of age 2. Notice that we cannot fire the transition t_e if there is more than one token in c_1 or c_2. Should M' be reachable, we have to fire another sequence of $t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6$. However, now there are at least two tokens of age 2 in c_r and all other tokens are of age 0. During firing of t_i^1, t_i^2, t_i^3 no time-elapsing is allowed (otherwise dead tokens appear). After t_i^4 is fired, there still remains at least one token of age 2 but there is no token of age 1 to enable t_i^5. This means that a time unit must pass in c_r to enable t_i^5, which causes that a dead token of age 3 appears in c_r. Thus whenever a 'cheating' sequence is fired, the marking M' cannot be reached. This means that the simulation is faithful and if $M \longrightarrow^* M'$ then R halts with both counters empty. □

Notice that the same construction works also for discrete time.

Theorem 3. *Reachability for LT nets is undecidable.*

On the other hand, it is sufficient to restrict the class of nets we consider very slightly in order to separate local and global timed nets.

Definition 8 (Safe marking and safe DTAPN).
A marking $M\colon P \to \mathcal{B}(D)$ is *safe* if $|M(p)| \leq 1$ for every $p \in P$. A marked DTAPN (N, M) is *safe* if the initial marking M is safe.

We now turn to show that the reachability problem for safe LT nets is decidable. Before proving the key decidability lemma, we fix some notation. For $N = (P, T, F, c, E, D)$ a DTAPN, let N^o denote the underlying ordinary Petri net (P, T, F). We define a time-forgetting function $f\colon [P \to \mathcal{B}(D)] \to \mathcal{B}(P)$, mapping DTAPN markings to PT net markings; the multiset $f(M)$ has precisely as many copies of p as there are numbers in $M(p)$. For $M^o \in \mathcal{B}(P)$ a marking of N^o, we use $\mathcal{T}(M^o) \subseteq 2^{[P \to \mathcal{B}(D)]}$ to denote the set of all timed-markings that have in each place the same number of tokens as M^o, i.e., $\mathcal{T}(M^o) = f^{-1}(M^o)$.

Lemma 3. *Let (N, M) be a safe marked LT net. If $f(M) \longrightarrow^* M_1^o$ in N^o then $M \longrightarrow^* M_1$ in N for every $M_1 \in \mathcal{T}(M_1^o)$.*

Proof. By induction on k we prove that if $f(M) \longrightarrow^k M_1^o$ in N^o then $M \longrightarrow^* M_1$ in N for every $M_1 \in \mathcal{T}(M_1^o)$.

Base case: If $k = 0$ then $M_1^o = f(M)$ and it can be easily seen that $M[\epsilon\rangle M_1$ for every $M_1 \in \mathcal{T}(f(M))$. Hence $M \longrightarrow M_1$ for every $M_1 \in \mathcal{T}(M_1^o)$.

Induction step: Let $k > 0$. Assume that $f(M) \longrightarrow^{k-1} M_1^o\ [t\rangle\ M_2^o$ in N^o. Let us fix an arbitrary $M_2 \in \mathcal{T}(M_2^o)$. We show that $M \longrightarrow^* M_2$ in N. By $\min_p$ we denote $\min\{M_2(p)\}$. We define a marking M_1 in N as follows:

$$M_1(p) = \begin{cases} M_2(p) & \text{if } p \in P \setminus ({}^\bullet t \cup t^\bullet) \\ M_2(p) \cup \{x\} & \text{if } p \in {}^\bullet t \setminus t^\bullet \text{ and } c(p,t) = (x, _) \\ \big(M_2(p) \dot{-} (-\min_p)\big) \setminus \{0\} & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ \big(M_2(p) \dot{-} (-\min_p)\big) \setminus \{0\} \cup \{x\} & \text{if } p \in {}^\bullet t \cap t^\bullet \text{ and } c(p,t) = (x, _). \end{cases}$$

Obviously, $M_1 \in \mathcal{T}(M_1^o)$. Because of induction hypothesis we know that $M \longrightarrow^* M_1$ in N. We prove our lemma by showing that $M_1 \longrightarrow^* M_2$. It is, however, easy to see that $M_1\ [t\rangle\ M_1'\ [\epsilon\rangle\ M_2$, where

$$M_1'(p) = \begin{cases} M_2(p) & \text{if } p \in P \setminus t^\bullet \\ M_2(p) \dot{-} (-\min_p) & \text{if } p \in t^\bullet \end{cases} \qquad \epsilon(\{p\}) = \begin{cases} 0 & \text{if } p \in P \setminus t^\bullet \\ \min_p & \text{if } p \in t^\bullet. \end{cases}$$

□

Theorem 4. *Reachability is decidable for safe LT nets, but undecidable for safe GT nets.*

Proof. Let (N, M) be a safe LT net. Trivially, for any marking M_1 reachable from M it is the case that $f(M_1)$ is reachable from $f(M)$ in N^o. Hence, using Lemma 3, the reachability problem for safe LT nets is reduced to the reachability problem for ordinary Petri nets, and this problem is decidable [10]. The reason for undecidability of reachability of safe GT nets is that in the undecidability proof from [15] the initial marking is safe. □

The coverability problem for GT nets was shown to be decidable — for discrete time in [14] and for continuous time in [1]. Following these results one proves that coverability is decidable even for DTAPNs.

Problem:	Coverability for DTAPNs.
Instance:	A marked DTAPN (N, M) and a final marking M'.
Question:	$\exists M''.\ M \longrightarrow^* M''\ \wedge\ \forall p \in P.\ M'(p) \subseteq M''(p)$?

Theorem 5. *Coverability for DTAPNs is decidable.*

5 Conclusion

We have studied a recently introduced model of distributed timed-arc Petri nets. The model is well motivated and captures e.g. the ideas behind the Globally

Asynchronous Locally Synchronous paradigm. We provide a formal process semantics for the model and compare the expressiveness of LT nets versus GT nets. We give answers to the most frequently studied decidability problems for Petri net models — reachability and coverability — finding a very delicate decidability borderline in the case of reachability for LT nets.

References

1. P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *Proc. of ICATPN 2001*, volume 2075 of *LNCS*, pages 53–70, 2001.
2. T. Bolognesi and P. Cremonese. The weakness of some timed models for concurrent systems. Technical Report CNUCE C89-29, CNUCE–C.N.R., 1989.
3. T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *Proc. of the IFIP WG 6.1 Tenth International Symposium on Protocol Spec., Testing and Verification*, pages 1–14, Amsterdam, 1990.
4. Fred D.J. Bowden. Modelling time in Petri nets. In *Proceedings of the Second Australia-Japan Workshop on Stochastic Models*, 1996. `http://www.itr.unisa.edu.au/~fbowden/pprs/stomod96/`.
5. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proc. of CAV'98*, volume 1427 of *LNCS*, pages 546–550, 1998.
6. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Computation*, 57:125–147, 1983.
7. H.M. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 282–299, 1993.
8. S. Hauck. Asynchronous design methodologies: An overview. In *Proc. of IEEE*, volume 83, pages 69–93, 1995.
9. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.
10. E.W. Mayr. An algorithm for the general Petri net reachability problem (preliminary version). In *Proc. of 13th Ann. ACM Symposium on Theory of Computing*, pages 238–246. Assoc. for Computing Machinery, 1981.
11. T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Berg, D. Lindqvist, H. Tenhunen, and A.Postula. Evaluating benefits of globally asynchronous locally synchronous VLSI architecture. In *Proceedings of 16th Norchip*, pages 50–57, 1998.
12. M. Nielsen, V. Sassone, and J. Srba. Towards a notion of distributed time for Petri nets. In *Proc. of ICATPN 2001*, volume 2075 of *LNCS*, pages 23–31, 2001.
13. C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets.* Ph.D. Thesis, Massachusetts Inst. of Tech., Cambridge, 1973.
14. V. Valero Ruiz, D. de Frutos Escrig, and O. Marroquin Alosno. Decidability of properties of timed-arc Petri nets. In Proc. of *ICATPN 2000*, volume 1825 of *LNCS*, pages 187–206, 2000.
15. V. Valero Ruiz, F. Cuartero Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proc. of PNPM'99*, pages 188–196, 1999.
16. J. Sifakis. Use of Petri nets for performance evaluation. In *Proc. of the 3rd International Symposium IFIP W.G. 7.3., Measuring, modelling and evaluating computer systems* , pages 75–93. Elsevier Science Publ., 1977.
17. J. Wang. *Timed Petri Nets, Theory and Application.* Kluwer Acad. Publ., 1998.

From Falsification to Verification

Doron Peled[1], Amir Pnueli[2], and Lenore Zuck[3]

[1] Univeristy of Texas at Austin, doron@ece.utexas.edu
[2] Weizmann Institute of Science, Rehovot, Israel, amir@wisdom.weizmann.ac.il
[3] New York University, New York, zuck@cs.nyu.edu

Abstract. This paper enhances the linear temporal logic model checking process with the ability to automatically generate a deductive proof that the system meets its temporal specification. Thus, we emphasize the point of view that model checking can also be used to *justify* why the system actually works. We show that, by exploiting the information in the graph that is generated during a failed search for counterexamples, we can generate a fully deductive proof that the system meets its specification.

1 Introduction

Model checking [1,2] is an automatic process for verifying temporal properties of finite state systems. Model checking techniques construct a (finite) model that represents the joint computations of the system and the negation of the property to be verified, and apply graph algorithmic techniques to the model to check that no computation of the system satisfies the negation of the property (and violates the property.) It was first applied to Linear Time Temporal Logic (LTL) properties in [5,6]. In the automata-theoretic approach ([11], and, independently, in [4]) both system and (negation of) property are explicitly represented as automata, and the intersection of the automata is checked for emptiness.

When executions that violate the property exist, at least one is reported and serves as a counterexample for the specification. When the search for counterexamples fails one may conclude that the system satisfies its specification. However, our confidence in such a positive conclusion is tarnished by two possible factors:

- For reasons of complexity and decidability, the system that is checked is often an oversimplification of the actual system, hence, failure to find counterexamples for the checked system does not necessarily imply a fault-free actual system, since faults of the actual system may have been abstracted away.
- The model checker itself may contain faults, causing it to report success when given a faulty system.

Both these risks may cause us to treat with diffidence a result that purely claims success without providing some supporting evidence–a "witness" or "certificate" that the property does indeed hold over the considered system. This 'proof by lack of counterexample' is the main drawback of the model checking approach;

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 292–304, 2001.
© Springer-Verlag Berlin Heidelberg 2001

some would even say that model checking is a tool for falsification rather than a tool for verification.

An alternative approach to model checking is *deductive verification* that incrementally constructs proofs until the desired conclusion, a proof the system meets its specification, is obtained. Deductive verification is often manual, and, like all deductive proofs, requires considerable human skill and time. One of its main benefits is that it often explains *why* the system satisfies its specification.

In this paper we enhance the LTL model checking process with the ability to automatically generate a deductive proof that the system meets its LTL specification. Thus, we emphasize the point of view that model checking can also be used to *justify* why the system actually works. We show that, by exploiting the information in the graph that is generated during a failed search for counterexamples, we can generate a fully deductive proof that the system meets its specification.

Several advantages are gained by a checker that, when the property is invalid, produces a counter example, and when the property is valid automatically produces a proof. For one, the proof can be independently checked by a theorem prover. (In fact, this can be used as a debugging tool for the model checker.) Moreover, if the system is a simplification of a more complex actual system, the proof can help to justify (or refute), and, in the case of justification, be transformed into a deductive proof of the property for the actual system.

The automata which represent the system and a tester for the LTL property, are represented in this paper as *Just Discrete Systems* (JDS) [3]. The JDS model, which is a variant of Fair Transition System model [8], is introduced in Section 2. A JDS is a transition systems that includes a set of *justice requirements*, each being an assertion that should be met by a computation of the system infinitely many times. JDSs correspond to Generalized Büchi automata in the automata theoretic view.

Both the system to be checked and the negation of the property are expressed by JDSs. We construct the synchronous parallel composition of these two JDSs to obtain a new JDS, the computations of which are the system computations that violate the property. If such a computation exists–the new JDS is *feasible*– it provides the desired counterexample. Otherwise, the graph generated by the composition is exploited to provide several alternative deductive proofs for the validity of the property over the system's computations.

The main challenge is how to represent the proof that is implicit in the composition JDS. We would like to represent it in a way that would explain to the user why the property holds for the checked system. We explore two alternatives of representing such a proof. Our first proof system (Section 3) automatically generates a proof based on well-founded ranking.

We then (Section 4) provide an algorithms that generates two temporal logic proof scripts, one that proves that the negation of the property is not satisfiable over the system, and the other proves that the property is valid over the system.

The process of generating the second proof script had been implemented in a new system, ProofProd, currently under construction. In Section 5 we present

a sample output of ProofProd. While the machine generated temporal logic proofs seems to be hard to read, minor heuristics can transform them to proofs that resemble efficient human-generated proofs.

Related Work: A preliminary version of this work [10] introduces the concept of generating the proof as a complementary stage of model checking using Generalized Büchi automata and general proof rules. In an independent work, Namjoshi [9] has shown a proof system for the μ-calculus, based on alternating tree automata and parity games. There, LTL is treated as a special case of CTL* using $\forall$-automata, and fairness is incorporated into the property.

2 Preliminaries

This section describes *Just Discrete Systems* (JSDs), the computational model we use here, reviews temporal logic, describes tableaux for JDSs, and shows how to use them for model checking.

2.1 Sequences

Let V be a set of typed variables. A V-state is an interpretation of V, assigning to each variable $x \in V$ a value in its respective domain. We denote by $s[x]$ the value s assigns to variable x. Let Σ_V denote the set of all V-states. A *sequence over* V, or a V*-sequence*, is a (possibly infinite) sequence $\sigma = s_0, s_1, \ldots$ over Σ_V. The *length of* σ, $|\sigma|$, is the number of states in a finite σ and ω otherwise.

Given a V-sequence $\sigma = s_0, s_1, \ldots$ and some i, $0 \leq i < |\sigma|$, we denote by σ^i the suffix of σ obtained by removing its first i elements. For a state assertion φ, we say that i *is a* φ*-position of* σ if s_i is a φ-state (i.e., $s_i \models \varphi$).

2.2 Just Discrete Systems

As a computational model for reactive systems, we take the model of *just* (*weakly fair*) *discrete system* (JDS), which is a variation of the model of *fair transition system* [8]. A JDS $\mathcal{D} : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J} \rangle$ consists of the following components:

- $V = \{u_1, ..., u_n\}$: A finite set of typed *system variables*, containing data and control variables. The set of *states* (interpretation) over V is denoted by Σ_V. When V is clear from the context, we denote Σ_V simply by Σ. Note that Σ can be both finite or infinite, depending on the domains of V.
- $\mathcal{O} = \{o_1, \ldots, o_n\} \subseteq V$: A finite set of *observable variables*. These are the variables which the environment can observe.
- Θ : The *initial condition* – an *assertion* (first-order state formula) characterizing the initial states.
- ρ : A *transition relation* – an assertion $\rho(V, V')$, relating the values V of the variables in state $s \in \Sigma$ to the values V' in a successor state $s' \in \Sigma$. For example, the assertion $x' = x + 1$ corresponds to the assignment $x := x + 1$.

  The state s' is defined to be a $\mathcal{D}$*-successor* of state s if $\langle s, s' \rangle \models \rho(V, V')$. That is, ρ evaluates to *true* when we interpret every $x \in V$ as $s[x]$ and every x' as $s'[x]$.

- $\mathcal{J}$: $\{J_1, \ldots, J_k\}$: A set of *justice* (*weak fairness*) *requirements*. The justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many J-states (states satisfying J).

We require that every state $s \in \Sigma$ has at least one $\mathcal{D}$-successor. This is often ensured by including in ρ the *idling* disjunct $V = V'$ (also called the *stuttering* step). In such cases, every state s is its own $\mathcal{D}$-successor.
A *computation* of $\mathcal{D} = \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J} \rangle$ is an infinite sequence of states

$$\sigma : s_0, s_1, s_2, ...,$$

satisfying the following requirements:

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...$, state s_{j+1} is an $\mathcal{D}$-successor of state s_j.
- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many J-positions

We denote by $\mathcal{C}omp(\mathcal{D})$ the set of computations of JDS $\mathcal{D}$.

A sequence satisfying the requirement of consecution is called a *run* of $\mathcal{D}$. A run satisfying the requirement of initiality is called an *initialized run*.

Let $U \subseteq V$ be a subset of the state variables and s be a V-state. We denote by $s\Downarrow_U$ the U-state, called the *projection of s on U*, which is obtained by restricting the interpretation of variables to the variables in U.

For a V-state sequence σ: $s_0, s_1, \ldots$, we denote by $\sigma\Downarrow_U$ the projected U-state sequence $\sigma\Downarrow_U$: $s_0\Downarrow_U, s_1\Downarrow_U, \ldots$. An $\mathcal{O}$-state sequence Ω is called an *observation* of the JDS $\mathcal{D}$ if $\Omega = \sigma\Downarrow_{\mathcal{O}}$ for some computation σ of $\mathcal{D}$. We denote by $Obs(\mathcal{D})$ the set of observations of JDS $\mathcal{D}$.

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *comparable* if they have the same sets of observable variables, i.e., $\mathcal{O}_1 = \mathcal{O}_2$. System $\mathcal{D}_2$, is said to be an *abstraction* of the comparable system $\mathcal{D}_2$, denoted $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$ if $Obs(\mathcal{D}_1) \subseteq Obs(\mathcal{D}_2)$. The comparable systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be *equivalent*, denoted $\mathcal{D}_1 \sim \mathcal{D}_2$, if their sets of observations are identical. That is, $Obs(\mathcal{D}_1) = Obs(\mathcal{D}_2)$. Note that if $\mathcal{D}_1$ and $\mathcal{D}_2$ are equivalent then each of them is an abstraction of the other.

A JDS $\mathcal{D}$ is said to be *feasible* if $\mathcal{D}$ has at least one computation. $\mathcal{D}$ is defined to be *viable* if any finite initialized run of $\mathcal{D}$ can be extended to a computation.

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *compatible* if $V_1 \cap V_2 = \mathcal{O}_1 \cap \mathcal{O}_2$.

The *synchronous parallel composition* of the compatible systems $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D}_1 ||| \mathcal{D}_2$, is given by the JDS $\mathcal{D} = \langle V_1 \cup V_2, W_1 \cup W_2, \mathcal{O}_1 \cup \mathcal{O}_2, \Theta_1 \wedge \Theta_2, \rho_1 \wedge \rho_2, \mathcal{J}_1 \cup \mathcal{J}_2 \rangle$, and is used for LTL model checking as shown below.

Claim. Let $\mathcal{D} = \mathcal{D}_1 ||| \mathcal{D}_2$. Then, a V-sequence σ is a computation of $\mathcal{D}$ iff $\sigma\Downarrow_{V_1}$ is a computation of $\mathcal{D}_1$ and $\sigma\Downarrow_{V_2}$ is a computation of $\mathcal{D}_2$. Similarly, the $\mathcal{O}$-sequence σ is an observation of $\mathcal{D}$ iff $\sigma\Downarrow_{\mathcal{O}_1}$ is an observation of $\mathcal{D}_1$ and $\sigma\Downarrow_{\mathcal{O}_2}$ is an observation of $\mathcal{D}_2$.

2.3 Temporal Logic

Let Π be a set of propositions, which can also be viewed as a set of boolean variables. A Π-state s is an interpretation of these variables, which we can

also represent as an element of 2^{Π}, i.e. a subset of Π, where $p \in s$ iff $s[p]$ is interpreted as 1 (true). We consider here linear time propositional temporal logic formulae over Π, using the Boolean connectives $\vee$ and $\neg$, and the temporal operators *next-time* $\bigcirc$ and *until* $\mathcal{U}$. Other Boolean connectives and Temporal operators ($\Box$, $\Diamond$, $\mathcal{V}$, etc.) can be defined using $\vee$, $\neg$, $\bigcirc$, and $\mathcal{U}$. Temporal logic formulae are interpreted over infinite sequences of Π-states in the usual way (see, e.g., [7].) Thus, $\sigma \models \varphi$ denotes that the infinite Π-sequence σ *satisfies* the temporal formula φ. Formula φ is said to be *satisfiable* is there exists a (Π-)sequence σ satisfying φ. Formula φ is *valid* if $\sigma \models \varphi$ for every Π-sequence σ.

Let P be a JDS with observables $\mathcal{O} = \Pi$ and φ be a temporal formula over Π. We say that φ is *valid over* P (P-*valid*) if every observation of P satisfies φ.

We assume that all the temporal formulae are given in the *positive normal form*, i.e., with negation applied only to propositions. This can be easily achieved by pushing negation inwards, using the following equivalences:

$$\neg\neg\varphi \equiv \varphi \qquad \neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi) \qquad \neg(\varphi \mathcal{U} \psi) \equiv (\neg\varphi)\, \mathcal{V}\, (\neg\psi)$$
$$\neg \bigcirc \varphi \equiv \bigcirc \neg\varphi \qquad \neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi) \qquad \neg(\varphi\, \mathcal{V}\, \psi) \equiv (\neg\varphi) \mathcal{U} (\neg\psi)$$

For an arbitrary formula φ, we denote by $pos(\varphi)$ the positive form formula which is equivalent to φ. We write $p \Rightarrow\!\!\!\!\!- q$ as an abbreviation of $\Box(p \rightarrow q)$.

2.4 Tableaux and Their Corresponding JDS's

Let φ be a temporal logic formula presented in positive normal form. Let *closure*(φ) be the set of formulae including all the sub-formulae of φ and the formulae $\bigcirc\psi$ for every $\mathcal{U}$ and $\mathcal{V}$ sub-formula ψ of φ. A φ-*atom* A is a (not necessarily maximal) logically consistent subset of *closure*(φ) that satisfies the following:

1. If A contains a conjunction $p \wedge q$ it must contain both p and q.
2. If A contains a disjunction $p \vee q$ it must contain p or q.
3. If A contains a formula $\psi = p \mathcal{U} q$, it must contain q, or both p and $\bigcirc\psi$.
4. If A contains $\psi = p\, \mathcal{V}\, q$, it must contain q and either p or $\bigcirc\psi$.

For an atom A, we denote by $\chi(A)$ the conjunction of all formulae contained in A. An *atom graph* $G_\varphi = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ for a formula φ consists of a set $\mathcal{A}$ of φ-atoms, a subset $\mathcal{A}_0 \subseteq \mathcal{A}$ of *initial atoms* such that $\varphi \in A$ for every $A \in \mathcal{A}_0$, and a set of edges $E \subseteq \mathcal{A} \times \mathcal{A}$ connecting atoms within the graph G. If $(A, B) \in E$, then it is required that $p \in B$ for every next-formula $\bigcirc p \in A$.
The atom graph $G = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ is said to be a *tableau* for φ if for every $A \in \mathcal{A}$:

$$\varphi \rightarrow \bigvee_{A \in \mathcal{A}_0} \chi(A) \qquad \textbf{and} \qquad \chi(A) \rightarrow \bigvee_{(A,B) \in E} \bigcirc \chi(B)$$

We refer to a proposition $p \in \Pi$ or a negation of a proposition as a *literal*. For an atom A, we denote by *prop*(A) the conjunction of all literals contained in A.

Let $G = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ be an atom graph, where $\mathcal{A} = \{A_1, \ldots, A_n\}$. We define $\mathcal{D}_G : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J} \rangle$, the JDS corresponding to G, by:

- $V = \Pi \cup \{\kappa : [1..n]\}$, i.e, V contains a control variable κ, ranging over $[1..n]$.

- $\mathcal{O} = \Pi$.
- Θ: $\bigvee_{A_i \in \mathcal{A}_0} (\kappa = i \ \wedge \ prop(A_i))$.
- ρ: $\bigvee_{(A_i, A_j) \in E} (\kappa = i \ \wedge \ \kappa' = j \ \wedge \ prop(A_j)')$.
- $\mathcal{J} = \{J_{p\,\mathcal{U}\,q} \mid p\mathcal{U}q \in \varphi\}$. Thus, $\mathcal{J}$ contains a justice requirement $J_{p\,\mathcal{U}\,q}$ for every *until* formula $p\mathcal{U}q$ contained in $closure(\varphi)$. The justice requirement $J_{p\,\mathcal{U}\,q}$ is given by

$$J_{p\,\mathcal{U}\,q}: \quad \Big(\bigvee_{q \in A_i} (\kappa = i)\Big) \ \vee \ \Big(\bigvee_{p\mathcal{U}q \notin A_i} (\kappa = i)\Big)$$

In the case that G is a tableau for the formula φ, we say that $\mathcal{D}_G$ is a *temporal tester* for φ and denote it by T_φ.

The following theorem summarizes the properties of a temporal tester:

Theorem 1. *Let T_φ be a temporal tester for the formula φ over the propositional variables Π. Then, a Π-sequence σ is an observation of T_φ iff $\sigma \models \varphi$. Furthermore, let $\sigma = s_0, s_1, \ldots$ be a computation of T_φ, let $i \geq 0$ be a position, and $j = s_i[\kappa]$ be the value of κ in state s_i. Then, $\sigma^i \models \chi(A_j)$.*

2.5 Model Checking

Let P be a finite-state JDS whose observables are Π. The goal of model checking is to establish the P-validity of a temporal property φ. This is accomplished by constructing a JDS whose observations are all the P-observations that satisfy $\neg\varphi$. If this JDS has no observations (equivalently, no computations), then φ is P-valid.

Let $\phi = \neg\varphi$, and let $T_\phi = T_{\neg\varphi}$ be a tester for ϕ. We define the JDS $\mathcal{D}_\phi^P = P \| \| T_\phi$. The following theorem follows immediately from Claim 2.2 and Theorem 1:

Theorem 2. *The formula φ is P-valid iff $\mathcal{D}_\phi^P$ is infeasible.*

Consequently, in order to verify that φ is P-valid, the process of model checking involves constructing the JDS $\mathcal{D}_\phi^P$ and checking that it is infeasible.

3 A Well-Founded Approach to *P*-Validity

Let φ be a temporal formula, P be a program, and $\phi = \neg\varphi$. We keep φ, ϕ, and P fixed for the sequel. We describe how to obtain a deductive proof of the P-validity of a property φ.

The JDS $\mathcal{D}_\phi^P$ can be viewed as a (labelled) directed graph $G_\phi^P = (S, S_0, \tau)$, where S is the set of states of $\mathcal{D}_\phi^P$, S_0 is the set of initial states, and τ is the set of edges connecting states to their immediate successors. We assume that every node in S is reachable from an initial state.

A *well-founded* domain $(\mathcal{W}, \succ)$ consists of a set $\mathcal{W}$ and a total ordering relation $\succ$ over $\mathcal{W}$ such that there are no infinitely decreasing chains $a_0 \succ a_1 \succ \ldots$. A *ranking* function for $\mathcal{D}_\phi^P$ is a function that maps the states of $\mathcal{D}_\phi^P$ into a well-founded domain. Assume that the justice requirements of $\mathcal{D}_\phi^P$ are $\{J_1, \ldots, J_r\}$. Rule WELL, presented in Fig. 1, can be used to prove that $\mathcal{D}_\phi^P$ is infeasible.

For a JDS $\mathcal{D}_\phi^P$ with justice requirements $J_1, \ldots, J_r$,
assertions $\varphi_1, \ldots, \varphi_r$
a well-founded domain $(\mathcal{W}, \succ)$,
and a ranking function $\delta \colon S \to \mathcal{W}$

W1. $\Theta \quad \rightarrow \bigvee_{j=1}^{r} \varphi_j$
W2. $\rho \ \wedge \ \varphi_i \quad \rightarrow (\varphi_i' \ \wedge \ \delta = \delta') \vee \bigvee_{j=1}^{r} (\varphi_j' \ \wedge \ \delta \succ \delta')$
W3. $\rho \ \wedge \ \varphi_i \ \wedge \ J_i' \quad \rightarrow \bigvee_{j=1}^{r} (\varphi_j' \ \wedge \ \delta \succ \delta')$

$\mathcal{D}_\phi^P$ is infeasible

Fig. 1. Rule WELL.

Theorem 3. WELL *is sound.*

Proof: It suffices to show that given $\varphi_1, \ldots, \varphi_r$, $(\mathcal{W}, \succ)$, and δ that satisfy the three premises W1-W3, $\mathcal{D}_\phi^P$ is infeasible. Assume to the contrary that $\mathcal{D}_\phi^P$ has a computation $\sigma \colon s_0, s_1, \ldots$. Since σ is a computation, $s_0 \models \Theta$. By premise W1, there exists some $i_0 \in \{1, \ldots, r\}$ such that $s_0 \models \varphi_{i_0}$. Denote $d_0 = \delta(s_0) \in \mathcal{W}$. By premises W2 and W3, there exists an $i_1 \in \{1, \ldots, r\}$ such that $s_1 \models \varphi_{i_1}$ and $d_1 = \delta(s_1) \preceq d_0$, where $d_0 = d_1$ implies $i_1 = i_0$ and $s_{i_1} \not\models J_{i_0}$. Proceeding in this way, we identify an infinite index sequence $i_0, i_1, \ldots$ and an infinite rank sequence $d_0 \succeq d_1 \succeq \ldots$ such that for every $j \geq 0$, $s_j \models \varphi_j$, $d_j = \delta(s_j)$, and $d_j = d_{j+1}$ implies that $i_j = i_{j+1}$ and $s_{j+1} \not\models J_{i_j}$. Since $\mathcal{W}$ is well-founded, the sequence of ranks cannot decrease infinitely many times. Thus, there exists some *stabilizing position* $c \geq 0$ such that for every $j \geq c$, $d_j = d_c$. It follows that for every $j \geq c$, $i_j = i_c$, and $s_{j+1} \not\models J_{i_c}$. Thus, σ violates the justice requirement J_{i_c} and is therefore not a computation of $\mathcal{D}_\phi^P$. □

We now describe how to automatically obtain the assertions, well-founded domain, and ranking functions: $\mathcal{W} = \mathbb{N}$ and $\succ$ is the usual $>$ ordering. Let T_ϕ^P be the dag obtained from G_ϕ^P after its separation to strongly connected components (SCCs). Each T_ϕ^P-node is either in some SCC of G_ϕ^P or *singular*, that is, not part of any SCC. For every SCC $\mathcal{C}$, let $\text{UNJUST}_\mathcal{C} \subseteq \{1, \ldots, r\}$ be the set of indices of the justice requirements that are not satisfied by any of $\mathcal{C}$'s nodes. For every $i = 1, \ldots, r$, let φ_i be a formula that describes the set of all singular nodes and nodes that belong to an SCC that violates J_i, that is, $\varphi_i = \{s : s \text{ is singular or } s \in \mathcal{C} \text{ s.t. } i \in \text{UNJUST}_\mathcal{C}\}$.

For all G^P_ϕ nodes s that are in T^P_ϕ's leaf nodes, $\delta(s) = 0$. If $\delta(s)$ is undefined, and δ is defined for all T^P_ϕ-successors $s_1, \ldots, s_k$ of s, then $\delta(s) = \max_{j=1}^{k} \delta(s_j)+1$.

Lemma 1 (Completeness). *If $\mathcal{D}^P_\phi$ is infeasible, then the procedure above produces assertions, well-founded domain, and ranking that satisfy the premises of* WELL.

Proof: Since $\mathcal{D}^P_\phi$ is infeasible, every SCC violates some J_i, and therefore any node that is in an SCC satisfies some φ_i. All singular nodes trivially satisfy all φ_is. Hence, every G^P_ϕ-state satisfies some φ_i. From the construction it is easy to see that every state leads either to a state with the same ranking (in the same SCC), or to a node with a lower ranking (out of the SCC), thus W2 holds. Assume that $s \models \varphi_i$, $s' \models J_i$, and $(s, s') \in \rho$. Since $s \models \varphi_i$, s belongs to an SCC neither of whose nodes satisfies J_i. Hence, s' and s are in different SCCs. Consequently, $\delta(s) > \delta(s')$, which establishes W3. □

4 From Falsification to Verification

This section presents a procedure that exploits the information in G^P_ϕ to generate deductive temporal proofs of φ's P-validity. In particular, we present an algorithm that generates simultaneously two proof scripts, one of $\mathcal{D}^P_\phi$'s infeasibility, and one of φ's P-validity. Thus, the first proof script establishes the falsification of ϕ, and the second establishes the verification of φ.

Let $s \in S$ be a state of $\mathcal{D}^P_\phi$, and assume that the state variables of P are $V^P = \{x_1, \ldots, x_\ell\}$. Let $\lambda(s) = \bigwedge_{i=1}^{\ell}(x_i = v_i)$. Note that the only V^P_ϕ-variable not included in V^P is κ. We denote by A_s the atom A_j where $j = s[\kappa]$ is the value s assigns to κ. We define

$$\mathcal{X}(s) = \lambda(s) \ \wedge \ \chi(A_s) \qquad \text{and} \qquad \bar{\chi}(s) = \neg\chi(A_s)$$

The formula $\mathcal{X}(s)$ is the *characteristic formula* of s. Its intended meaning is to describe the temporal formulae hold when a computation of $\mathcal{D}^P_\phi$ reaches s. The formula $\bar{\chi}(s)$ describes the temporal formulae that hold when an initial run of an infeasible $\mathcal{D}^P_\phi$ reach s. Note that $\mathcal{X}(s)$ does not refer to the state variable κ. However, since $\chi(A_j)$ uniquely identifies κ as having the value j, this is not necessary.

The properties of the characteristic formulae and of the $\bar{\chi}$s are stated in the following claim, whose proof follows immediately from the definitions and from Theorem 1:

Lemma 2. *For every computation $\sigma = s_0, \ldots$ of $\mathcal{D}^P_\phi$, and every $i \geq 0$, $\sigma^i \models \mathcal{X}(s_i)$. Similarly, if $\mathcal{D}^P_\phi$ is infeasible, then for every run $\sigma = s_0, \ldots$ of $\mathcal{D}^P_\phi$, and every $i \geq 0$, $\sigma^i \models \bar{\chi}(s_i)$*

An immediate corollary of Lemma 2 is:

Corollary 1. *$\mathcal{D}_\phi^P$ is infeasible iff $\mathcal{X}(s_0) \to \text{F}$ for every $s_0 \in S_0$. Similarly, φ is P-valid iff $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$ for every $s_0 \in S_0$.*

Based on Corollary 1, we describe a procedure that attempts to show that $\mathcal{X}(s) \Rightarrow \text{F}$ (for the first proof script) and that $\lambda(s) \Rightarrow \bar{\chi}(s)$ (for the second proof script) for every state $s \in S$. If φ is P-valid, our procedure terminates after showing the above for every initial state $s_0 \in S_0$, which, by Corollary 1, achieves our goal. If φ is not P-valid (and $\mathcal{D}_\phi^P$ is feasible) our procedure terminates while failing to show the above for every $s_0 \in S_0$.

Both proofs are achieved by a chain of temporal formulae that are P-valid, and each either follows immediately from the properties of $\mathcal{D}_\phi^P$, or from the previous P-validates in the chain. Initially, the proof is empty. It then proceeds according to the procedure in Figure 2.

The procedure in Figure 2 describes how both proof scripts are constructed. The procedure resembles the completeness proof of [5,6], but, while the procedure there is purely semantic (working on the graph), here we give it a syntactic flavor.

(a) For each program justice requirement $J \in \mathcal{J}^P$, add to the proof the line
$\Box \Diamond J$

(b) For every state $s \in S$, add to the proof the line
$\mathcal{X}(s) \Rightarrow \bigvee_{s' \in \tau(s)} \bigcirc \mathcal{X}(s')$

(c) Let all S's nodes be "unmarked". Repeat the following until all nodes are marked:

(c.1) If there is an unmarked node all of whose successors are marked, then mark s and to the proof add the line
$\mathcal{X}(s) \Rightarrow \text{F}$ | $\lambda(s) \Rightarrow \bar{\chi}(s)$

(c.2) If there is an SCC $\mathcal{C}$ of $\mathcal{D}_\phi^P$, all of whose nodes are unmarked, such that all exits from $\mathcal{C}$ lead to marked nodes and for some justice requirement $J \in \mathcal{J}$ no state $\mathcal{C}$ is a J-state, then for every node $s \in \mathcal{C}$ mark s and add to the proof the line(s)
$\mathcal{X}(s) \Rightarrow \Box \bigvee_{s' \in \mathcal{C}} \mathcal{X}(s')$ | $\lambda(s) \Rightarrow \bar{\chi}(s)$
$\mathcal{X}(s) \Rightarrow \text{F}$ |

(c.3) If no node was marked in either (c.1) or (c.2), report a counter-example and halt.

Fig. 2. A Procedure to Generate a Temporal Proof of P-validity

Theorem 4. *If the procedure of Figure 2 terminates with all nodes marked , then for every initial node s_0, $\mathcal{X}(s_0) \Rightarrow \text{F}$ and $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$. Moreover, for every initial node s_0, the formulae generated by the first proof script constitute a proof that $\mathcal{X}(s_0) \Rightarrow \text{F}$, and the formulae generated by the second proof script constitute a proof that $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$.*

Proof Outline: Note first that the procedure marks all nodes only if there is no path in $\mathcal{D}_\phi^P$ leading from an initial node to a SCC that satisfies every justice

requirement. Thus, the procedure terminates with all nodes marked only when $\mathcal{D}_\phi^P$ is infeasible.

The validity of steps (a) and (b) is immediate. In step (c.1) we are considering (and marking, if possible) a node that does not belong to any SCC from the graph. That is, we are removing state s which, at the time of removal, has all its successors marked. Assume that the successors of s in the original graph G_ϕ^P are $s_1, \ldots, s_k$. In step (b) we added to the proof the lines

$$\mathcal{X}(s) \;\Rightarrow\; \bigvee_{i=1}^{k} \bigcirc \mathcal{X}(s_i) \tag{1}$$

Since all of the successors are marked, it follows that for each $i = 1, \ldots, k$, the first and second proofs contains the lines $\mathcal{X}(s_i) \Rightarrow \text{F}$ and $\lambda(s_i) \Rightarrow \bar{\chi}(s_i)$ respectively. Combining these lines with formula (1), we conclude $\mathcal{X}(s) \;\Rightarrow\; \text{F}$ and $\lambda(s) \;\Rightarrow\; \bar{\chi}(s)$.

In step (c.2) we are dealing with an SCC $\mathcal{C}$, all of whose nodes are unmarked, and all of whose exit edges lead to marked nodes. Consider a state $s \in \mathcal{C}$. Viewing again proof line (b) for state s, we observe that for every immediate successor s' of s which lies outside of $\mathcal{C}$ the proofs contain lines $\mathcal{X}(s') \Rightarrow \text{F}$ and $\lambda(s') \Rightarrow \bar{\chi}(s')$ respectively. Consequently, we can reduce formula (1) into the formula $\mathcal{X}(s) \;\Rightarrow\; \bigvee_{(s,s') \in \tau, s' \in \mathcal{C}} \bigcirc \mathcal{X}(s')$, which in turn can be weakened into $\mathcal{X}(s) \;\Rightarrow\; \bigvee_{s' \in \mathcal{C}} \mathcal{X}(s')$. Taking the disjunction of the above over all $s \in \mathcal{C}$, we obtain $\bigvee_{s \in \mathcal{C}} \mathcal{X}(s) \;\Rightarrow\; \bigcirc \left(\bigvee_{s \in \mathcal{C}} \mathcal{X}(s)\right)$, from which, by the axioms of LTL, we can infer that for every $s \in \mathcal{C}$,

$$\mathcal{X}(s) \;\Rightarrow\; \Box\Big(\bigvee_{s \in \mathcal{C}} \mathcal{X}(s)\Big) \tag{2}$$

There are two possible reasons why SCC $\mathcal{C}$ was marked. Either it failed to satisfy one of the program justice requirements $J \in \mathcal{J}^P$, or one of the atoms A_s for some $s \in \mathcal{C}$ contained the formula $p \mathcal{U} q$ and no state within $\mathcal{C}$ satisfies q.

In the first case, the failure to satisfy J implies that $\mathcal{X}(s) \rightarrow \neg J$ for every $s \in \mathcal{C}$. By formula (2) this implies $\mathcal{X}(s) \Rightarrow \Box \neg J$ which, in view of the line $\Box \diamondsuit J$ originally placed in the proof leads to $\mathcal{X}(s) \Rightarrow \text{F}$, however, since $\mathcal{X}(s) \leftrightarrow \lambda(s) \wedge \neg \bar{\chi}(s)$, it follows that $\lambda(s) \Rightarrow \bar{\chi}(s)$.

Consider the second case in which all atoms within $\mathcal{C}$ contain the formula $p \mathcal{U} q$ but do not contain the formula q. Let s be a node within $\mathcal{C}$. Since $\mathcal{C}$ is strongly connected, s must have an immediate predecessor $\widehat{s} \in \mathcal{C}$. Let $s_1, \ldots, s_k$ be all the nodes which are immediate successors of $\widehat{s}$ such that $q \in A_{s_i}$ for every $i = 1, \ldots, k$. From the construction it can be established that

$$\mathcal{X}(s) \;\wedge\; q \Rightarrow \bigvee_{i=1}^{k} \mathcal{X}(s_i) \tag{3}$$

Since no node within $\mathcal{C}$ contains q in its atom, it follows that $s_1, \ldots, s_k$ are outside of $\mathcal{C}$ and, as $\mathcal{C}$ is currently marked, the proofs must contain lines $\mathcal{X}(s_i) \Rightarrow \mathrm{F}$ and $\lambda(s_i) \Rightarrow \bar{\chi}(s_i)$ respectively, for every $i = 1, \ldots, k$. Together with formula (3), this implies $\mathcal{X}(s) \ \wedge \ q \Rightarrow \mathrm{F}$. It follows that $\mathcal{X}(s) \Rightarrow (p \mathcal{U} q \ \wedge \ \neg q)$ for every $s \in \mathcal{C}$. By formula (2) this implies $\mathcal{X}(s) \Rightarrow \Big(\Box(p \mathcal{U} q) \ \wedge \ \Box \neg q\Big)$ which is equivalent to $\mathcal{X}(s) \Rightarrow \mathrm{F}$ and to $\lambda(s) \Rightarrow \bar{\chi}(s)$. □

5 Experimental Results

We have constructed ProofProd, a prototype system that generates temporal proofs as described above. For example, we ran ProofProd on the following JDS P: V consists of $\mathtt{L} \in \{0, 1, 2\}$ and $p \in \{\mathrm{T}, \mathrm{F}\}$. We use $\mathtt{Lj}$ to denote $\mathtt{L} = j$. Θ is $\mathtt{L0} \wedge p$; The single justice requirement is $J = \mathtt{L0} \vee \mathtt{L2}$. The transition relation ρ is given by: $(\mathtt{L0} \wedge \mathtt{L0}' \wedge p') \vee ((\mathtt{L0} \vee \mathtt{L1}) \wedge \mathtt{L1}' \wedge \neg p') \vee ((\mathtt{L1} \vee \mathtt{L2}) \wedge \mathtt{L2}' \wedge p')$. The property whose P-validity we'd like to establish is $\varphi = \ \diamondsuit\Box p$. The atom graph for $\phi = \neg\varphi = \Box \ \diamondsuit \neg p$ is described in Fig. 3. There, source-less edges point to initial states, and double circles denote states belonging to the (single) justice set J_ϕ.

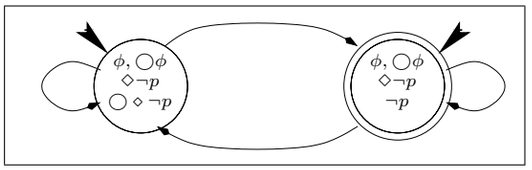


Fig. 3. An atom graph of T_ϕ

Fig. 4 contains the proof that ProofProd generated where we used an option that displays the proof in LaTeX. We are currently working on heuristics to simplify temporal logic formulae that will allow to make the proof more compact. Fairly simple transformations can establish that all of $\Box p$, $\bigcirc \ \diamondsuit\Box p$, and $\bigcirc\Box p$ imply $\varphi = \ \diamondsuit\Box p$, which helps simplify lines(6)–(9) of the proof in Fig. 4 to

$(6')\ (\mathtt{L2} \wedge p) \rightarrow \varphi \quad (7')\ (\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee p) \quad (8')\ (\mathtt{L1} \wedge \neg p) \rightarrow \varphi \quad (9')\ (\mathtt{L0} \wedge p) \rightarrow \varphi$

(1) $\Box\diamond(\mathtt{L0} \vee \mathtt{L2})$
(2) $(\mathtt{L0} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \rightarrow \bigcirc((\mathtt{L0} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \vee (\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \neg p))$
(3) $(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \rightarrow \bigcirc((\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \neg p) \vee (\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p))$
(4) $(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \neg p) \rightarrow \bigcirc((\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \neg p) \vee (\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p))$
(5) $(\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p) \rightarrow \bigcirc((\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \diamond\neg p \wedge \bigcirc\diamond\neg p))$
(6) $(\mathtt{L2} \wedge p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$
(7) $(\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee p)$
(8) $(\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$
(9) $(\mathtt{L0} \wedge p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$

Fig. 4. Proof script of P-validity of φ

Another set of heuristics is more specific to the type of formulae we obtain in the proof script. Consider the procedure of Figure 2. For every SCC $\mathcal{C}$, let $exit(\mathcal{C}) = \{s' : \text{for some } (s, s') \in \tau, s \in \mathcal{C} \text{ and } s' \notin \mathcal{C}\}$ i.e., $exit(\mathcal{C})$ is the set of all nodes the resider directly outside of $\mathcal{C}$. It is easy to see that, if the procedure terminates when all nodes are marked, then for every SCC C that is marked in step 3.2 because if fails to satisfy a program justice requirement,

$$\lambda(s) \Rightarrow (\bigvee_{t \in \mathcal{C}} \lambda(t) \ \wedge \ \bar{\chi}(t)) \mathcal{U} (\bigvee_{t \in exit(\mathcal{C})} \bar{\chi}(t)) \tag{4}$$

Similarly, for every SCC C that is marked in step 3.2 because if fails to satisfy a $p \mathcal{U} q$ justice requirement,

$$\lambda(s) \Rightarrow (\bigvee_{t \in \mathcal{C}} \lambda(t) \ \wedge \ \bar{\chi}(t)) \mathcal{W} (\bigvee_{t \in exit(\mathcal{C})} \bar{\chi}(t)) \tag{5}$$

In our example, since the SCC that consists of the single node whose λ is $\mathtt{L2} \wedge p$ is removed because if fails to satisfy the justice requirement $\diamondsuit\neg p$, and its set of exit nodes is empty, we obtain from formula (5) that $\mathtt{L2} \wedge p \Rightarrow (\mathtt{L2} \wedge p \ \wedge \ (\varphi \vee p)) \mathcal{W}$ F, which implies (by LTL theorems) and the simplification rules above that $\mathtt{L2} \wedge p \Rightarrow \Box(\mathtt{L2} \wedge p)$. Using a simplification rule "$\Box(p \wedge q) \rightarrow \Box p$", we can obtain $\mathtt{L2} \wedge p \Rightarrow \Box p$ instead of line (6) in the proof.

These, and other heuristics we are working on, help to greatly simplify the proof scripts generated by ProofProd, making the proof methodology advocated here both practical and beneficial.

6 Conclusion and Future Work

The paper demonstrates how model-checking, that is considered useful only for purposes of falsification, can be used to obtain deductive verification.

As reported in Section 5, we are currently working on heuristics to our system that will generate proofs that are closer to "human" proofs. We are also working on extending our results to apply to systems that employ a wider set of fairness constraints, as well as on obtaining deductive proofs from symbolic model checkers.

Acknolwedgement We would like to thank Yi Fang for her technical assistance with ProofProd, and thank Jessie Xu and Yonit Kesten for careful proofreading of the manuscript.

References

1. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, LNCS 131, 1981.

2. E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In ICALP'80, pp. 169–181, LNCS 85.
3. Y. Kesten and A. Pnueli. Verification by finitary abstraction. *Information and Computation*, 163:203–243, 2000.
4. R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, New Jersey, 1995.
5. O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In POPL'85, pp. 97–107, 1985.
6. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In LICS'85, LNCS 193, pp. 196–218, 1985.
7. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
8. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
9. K.S. Namjoshi. Certifying model checkers. In CAV'01, LNCS 2102, pp. 2–13.
10. D. Peled and L. Zuck. From model checking to a temporal proof. In SPIN'2001, LNCS 2057 pp. 1–14.
11. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In LICS'86, pp. 332–344, 1986.

On Polynomial Representations of Boolean Functions Related to Some Number Theoretic Problems

Erion Plaku[1,*] and Igor E. Shparlinski[2,**]

[1] Department of Mathematics and Computer Science
Clarkson University, Potsdam, NY 13699-5815, USA
plakue@clarkson.edu
[2] Department of Computing,
Macquarie University, NSW 2109, Australia
igor@ics.mq.edu.au

Abstract. We say a polynomial P over $\mathbb{Z}_M$ *strongly* M-represents a Boolean function F if $F(x) \equiv P(x) \pmod M$ for all $x \in \{0,1\}^n$. Similarly, P *one-sidedly* M-represents F if $F(x) = 0 \iff P(x) \equiv 0 \pmod M$ for all $x \in \{0,1\}^n$. Lower bounds are obtained on the degree and the number of monomials of polynomials over $\mathbb{Z}_M$, which strongly or one-sidedly M-represent the Boolean function deciding if a given n-bit integer is square-free. Similar lower bounds are also obtained for polynomials over the reals which provide a threshold representation of the above Boolean function.

1 Introduction

In this paper, we obtain lower bounds on the degree and the number of monomials of polynomials over $\mathbb{Z}_M$, which strongly or one-sidedly M-represent the Boolean function deciding if a given n-bit integer is square-free. These results provide the first non-trivial lower bounds over $\mathbb{Z}_M$ on the complexity of a number theoretic problem which is closely related to the integer factorization problem. Similar lower bounds are also obtained for polynomials over the reals which provide a threshold representation of the above Boolean function.

We also show that some simple number theoretic observations allow us to obtain quite strong lower bounds on several other complexity characteristics of testing if a given integer is square-free.

We recall that an integer x is called *square-free* if there is no prime p such that $p^2|x$. Otherwise, x is called *square-full*. We define the function

$$S(x) = \begin{cases} 1, & \text{if } x \text{ is square-free,} \\ 0, & \text{if } x \text{ is square-full.} \end{cases}$$

* Supported in part by NSF grant CCR-9877150.
** Supported in part by ARC grant A69700294.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 305–316, 2001.
© Springer-Verlag Berlin Heidelberg 2001

For a given integer $n \ge 1$, we can identify x, $0 \le x \le 2^n - 1$, and its bit representation $x_1 \ldots x_n$ (if necessary we add several leading zeroes) and consider $S(x)$ as a Boolean function of n variables.

We say a polynomial P over $\mathbb{Z}_M$ *strongly* M-represents S if for all $1 \le x \le 2^n - 1$,

$$P(x_1, \ldots, x_n) \equiv S(x) \pmod{M}, \tag{1}$$

where $x = x_1 \ldots x_n$ is the bit representation of x.

Similarly, we say a polynomial P over $\mathbb{Z}_M$ *one-sidedly* M-represents S if for all $1 \le x \le 2^n - 1$,

$$P(x_1, \ldots, x_n) \equiv 0 \pmod{M} \iff S(x) = 0, \tag{2}$$

where $x = x_1 \ldots x_n$ is the bit representation of x.

For Boolean inputs we simply need to consider multilinear polynomials. Each polynomial over $\mathbb{Z}_M$ is of the form

$$P(x_1, \ldots, x_n) = \sum_{H \in \mathcal{H}} A_H \prod_{i \in H} x_i, \tag{3}$$

where

$$\mathcal{H} \subseteq 2^{\{1,2,\ldots,n\}} \quad \text{and} \quad 0 \ne A_H \in \mathbb{Z}_M.$$

We call the largest value of $|H|$ in the representation (3) the *degree* of P and write $\deg P$. We call the number of coefficients A_H, or equivalently $|\mathcal{H}|$, the *sparsity* of P and write $\operatorname{spr} P$.

In this paper, we obtain lower bounds on the degree $\deg P$ and the sparsity $\operatorname{spr} P$ of polynomials over $\mathbb{Z}_M$, satisfying either (1) or (2) for all inputs.

Similarly to the case of polynomials over $\mathbb{Z}_M$, for a polynomial f in n variables over the reals $\mathbb{R}$, we define the total degree $\deg f$ as the largest sum $i_1 + \ldots + i_n$ and the sparsity $\operatorname{spr} f$ as the number of coefficients $A_{i_1 \ldots i_n}$ in the representation

$$f(x_1, \ldots, x_n) = \sum_{i_1, \ldots, i_n} A_{i_1, \ldots, i_n} x_1^{i_1} \ldots x_n^{i_n}, \qquad A_{i_1, \ldots, i_n} \ne 0.$$

For a real w we define the sign-function as

$$\operatorname{sign} w = \begin{cases} 1, & \text{if } w \ge 0, \\ 0, & \text{if } w < 0. \end{cases}$$

Here we also obtain lower bounds on the degree $\deg f$ and sparsity $\operatorname{spr} f$ of polynomials f providing a *threshold* representation of $S(x)$ for n-bit integers x, that is a representation of the form

$$\operatorname{sign} f(x_1, \ldots, x_n) = S(x),$$

where $x = x_1 \ldots x_n$ is the bit representation of x, $1 \le x \le 2^n - 1$.

Furthermore, in the case of real polynomials, the Boolean values 0 and 1 can be interpreted as two arbitrary real values α_0 and α_1, not necessarily $\alpha_0 = 0$ and

$\alpha_1 = 1$. It is easy to see that the degree of the corresponding polynomials does not depend on the particular choice of α_0, α_1 because they are equivalent under a linear transformation of variables [19]. But it is shown in [19] that the sparsity $\operatorname{spr} f$ depends on the choice of α_0 and α_1. In fact, there are examples of Boolean functions demonstrating that for $(\alpha_0, \alpha_1) = (0, 1)$ and $(\alpha_0, \alpha_1) = (1, -1)$ the gap between the numbers of monomials of the corresponding polynomials for these two representations can be exponentially large [19].

Threshold representations of Boolean functions via real polynomials have been studied in a number of works [8,9,14,19,24,28]. These papers contain many general estimates together with lower bounds for some particular Boolean functions. However, these Boolean functions are usually specially constructed examples which are not related to any particular number theoretic or combinatorial problem.

Representations of Boolean functions via polynomials over $\mathbb{Z}_M$ have been studied in [2,3,15,30]. In these papers, lower and upper bounds are obtained for polynomials representing the OR, MOD_M (that determines if the sum of the inputs is not divisible by M), and $\neg\mathrm{MOD}_M$ Boolean functions. We note that a polynomial of degree d over $\mathbb{Z}_M$ is represented by a circuit consisting of an unbounded fan-in MOD_M gate at the top where each input wire is a function of no more than d variables. In [12,29], some lower bounds are obtained for polynomials over $\mathbb{Z}_2$ strongly 2-representing the Boolean function deciding the quadratic residuacity of an n-bit integer x.

In the series of papers [4,5,6,7] lower bounds have been obtained on the circuit complexity, sensitivity, degree of polynomial representation and other complexity characteristics of testing square-free numbers and computing the greatest common divisor. As in [12,29] the method of [4,5,6,7] is based on the uniformity of distribution of long patterns of $0, 1$ in the values of $S(x)$. For the quadratic residuacity a similar property has been established in [12,29] by using the very powerful Weil estimate, in [4,5,6,7] a sieve method has been used for this purpose. In particular, for a strongly 2-representing polynomial P the lower bound

$$\deg P \geq 0.165\ldots\, n$$

has been obtained in [5]. It has also been applied to obtain a lower bound of order $n^{1/2}$ on the degree of real polynomials P which approximate S in the following sense: for all $1 \leq x \leq 2^n - 1$,

$$|S(x) - P(x_1, \ldots, x_n)| \leq 1/3$$

where $x = x_1 \ldots x_n$ is the bit representation of x. These lower bounds are derived from the asymptotic formula for the sensitivity of the function S obtained in [5]. Unfortunately, there is no link between the sensitivity and the degrees of M-representing polynomials, $M \geq 3$, and of threshold representations.

Alternative methods of [1] and [32] yield stronger but less explicit complexity results (which apply to primality testing as well). However these approaches work neither for M-representing polynomials nor for threshold representations.

Here we use the technique of [4,5,6,7] to obtain several new results about polynomial representation of the function $S(x)$.

Throughout the paper we denote by $\log x$ the binary logarithm of x, by $\ln x$ the natural logarithm of x, and $\exp(x) = e^x$.

2 Auxiliary Results

Let $\mathcal{P}$ denote the set of primes.

We use the following well known asymptotic formulas (see [13] for example)

$$\ln\left(\prod_{\substack{p \le x \\ p \in \mathcal{P}}} p\right) = x + O\left(\frac{x}{\ln x}\right), \qquad x \to \infty. \tag{4}$$

and

$$\pi(x) = \frac{x}{\ln x} + O\left(\frac{x}{\ln^2 x}\right), \qquad x \to \infty, \tag{5}$$

for the number of primes $p \le x$.

The following estimate can be found in [20], Section 10.11.

Lemma 1. *For any integers L and N with $0 \le L < N/2$ the bound*

$$\sum_{K=0}^{L} \binom{N}{K} \le 2^{H(L/N)N}$$

holds, where $H(\gamma) = -\gamma \log \gamma - (1-\gamma)\log(1-\gamma), \quad 0 < \gamma < 1$, is the binary entropy function.

Now we prove the following quite technical statement.

Lemma 2. *Let $m \ge 1$ be an integer and let us define k from the inequalities*

$$2^k \ge m^2 > 2^{k-1}.$$

Let $m < p_1 < \ldots < p_m$ be the first m primes which are greater than m. Then, for any m-dimensional binary vector $(\sigma_1, \ldots, \sigma_m)$ exists an integer y, such that $0 \le y \le \exp\left(4m \ln m + O(m \ln \ln m)\right)$ and

$$S(2^k y + p_i) = \sigma_i, \qquad i = 1, \ldots, m.$$

Proof. Put

$$Q = \prod_{\substack{p \le m \\ p \in \mathcal{P}}} p \qquad \text{and} \qquad R = 2^k Q.$$

From (4) we see that $Q = \exp\left(O(m)\right)$. Thus it is enough to show that there exists an integer u such that $0 \le u \le \exp\left(4m \ln m + O(m \ln \ln m)\right)$ and

$$S(Ru + p_i) = \sigma_i, \qquad i = 1, \ldots, m. \tag{6}$$

We remark that $\gcd(p_i, R) = 1$, $i = 1, \ldots, m$.

Let $\mathcal{I}$ be the set of subscripts i for which $\sigma_i = 0$ and let $\mathcal{J}$ be the set of subscripts j for which $\sigma_j = 1$. Put

$$q = \prod_{i \in \mathcal{I}} p_i^2.$$

From the Chinese Remainder Theorem we conclude that there exists an integer a, $0 \le a \le q - 1$, such that $Ra \equiv -p_i \pmod{p_i^2}$, for all $i \in \mathcal{I}$. Therefore, $R(qz + a) + p_i \equiv 0 \pmod{p_i^2}$, for all $i \in \mathcal{I}$ and any integer z. Now we show that one can select a not too large z for which

$$S\left(R(qz + a) + p_j\right) = 1, \qquad j \in \mathcal{J}.$$

For $Z \ge 1$, we denote by $L_j(Z)$ the number of square-full numbers of the form $R(qz + a) + p_j$ with $1 \le z \le Z$, $j \in \mathcal{J}$. To prove the lemma it is sufficient to show that for some appropriate Z,

$$\sum_{j \in \mathcal{J}} L_j(Z) < Z. \tag{7}$$

First of all, we remark that, for $i \in \mathcal{I}$ and $j \in \mathcal{J}$,

$$R(qz + a) + p_j \not\equiv 0 \pmod{p_i^2}.$$

Otherwise, we have $p_i^2 | (p_j - p_i)$ which is impossible.

For any prime $p \in \mathcal{P}$ with $\gcd(p, q) = 1$, the congruence

$$R(qz + a) + p_j \equiv 0 \pmod{p^2}, \qquad 1 \le z \le Z,$$

has at most $Z/p^2 + 1$ solutions. Obviously, it does not have solutions for $p^2 > Rq(Z + 1) + R$. Put $V = (3RqZ)^{1/2}$.

The smallest prime divisor of any number $R(qz + a) + p_j$ exceeds m. Therefore,

$$\begin{aligned}
L_j(Z) &\le \sum_{\substack{m < p \le V \\ \gcd(p,q)=1}} \left(\frac{Z}{p^2} + 1\right) \le Z \sum_{p > m} \frac{1}{p^2} + O\left(\frac{V}{\ln V}\right) \\
&\le Z \sum_{\nu \ge \lfloor \log m \rfloor} \sum_{2^{\nu+1} > p \ge 2^\nu} \frac{1}{p^2} + O\left(\frac{V}{\ln V}\right) \\
&\le Z \sum_{\nu \ge \lfloor \log m \rfloor} \frac{\pi(2^{\nu+1})}{2^{2\nu}} + O\left(\frac{V}{\ln V}\right) \\
&\le O\left(Z \sum_{\nu \ge \lfloor \log m \rfloor} \frac{1}{2^\nu \nu} + \frac{V}{\ln V}\right) = O\left(\frac{Z}{m \ln m} + \frac{V}{\ln V}\right).
\end{aligned}$$

Putting $Z = m^2 Rq$ we obtain the inequality (7), provided that m is large enough. Therefore, there exists an integer u satisfying condition (6) and $0 \le u \le q(Z + 1) \le 2m^2 Rq^2$.

Now, from (5) we conclude that $p_m = m \ln m + O(m)$. Therefore, we have $q \le \exp(2m \ln m + O(m \ln \ln m))$. Finally, from (4) we see that $R = \exp(O(m))$, and the result follows. □

The result of Lemma 2 can be improved by means of some more sophisticated sieve methods, see [17] for example. However, this does not improve our main results.

3 Main Results

First of all we consider deciding the property of being square-free via polynomials in $\mathbb{Z}_M[X_1, \ldots, X_n]$.

Theorem 1. *Assume that a polynomial*

$$P(X_1, \ldots, X_n) \in \mathbb{Z}_M[X_1, \ldots, X_n]$$

strongly M-represents $S(x)$, *that is, it is such that for any* x, $1 \le x \le 2^n - 1$,

$$P(x_1, \ldots, x_n) \equiv S(x) \pmod{M},$$

where $x = x_1 \ldots x_n$ *is the bit representation of* x. *Then, for sufficiently large* n, *the bounds*

$$\deg P \ge 0.14 \ln n \qquad \textit{and} \qquad \operatorname{spr} P \ge \frac{n}{5 \ln n}$$

hold.

Proof. Assuming that n is large enough, we put

$$m = \left\lceil \frac{n}{5 \ln n} \right\rceil.$$

Let $p_1, \ldots, p_m$ and k be defined as in Lemma 2.

We denote by τ the number of monomials $\mu_j(w)$, $j = 1, \ldots, \tau$, in $w = (w_1, \ldots, w_k)$, such that for every k-dimensional vector

$$w = (w_1, \ldots, w_k) \in \{0, 1\}^k$$

we have a representation of the form

$$P(Y_1, \ldots, Y_{n-k}, w) = \sum_{j=1}^{\tau} \mu_j(w) f_j(Y_1, \ldots, Y_{n-k})$$

with some polynomials $f_j(Y_1, \ldots, Y_{n-k}) \in \mathbb{Z}_M[Y_1, \ldots, Y_{n-k}]$.

Obviously,

$$\tau \le \sum_{l=0}^{\deg P} \binom{k}{l} \qquad \text{and} \qquad \tau \le \operatorname{spr} P. \tag{8}$$

As in the proof of Lemma 2, we note that $p_1 < \ldots < p_m < m^2 \le 2^k$. For every $i = 1, \ldots, m$, we add several leading zeroes to the binary representation of p_i to obtain binary strings s_i of length k.

If $\tau < m$, then there exist m integer coefficients $c_1, \ldots, c_m$, not all equal to zero, with

$$\sum_{i=1}^{m} c_i \mu_j(s_i) = 0, \qquad j = 1, \ldots, \tau.$$

Therefore we have the identity:

$$\sum_{i=1}^{m} c_i P(X_1, \ldots, X_{n-k}, s_i) = 0.$$

Without loss of generality we can also assume that

$$\gcd(c_1, \ldots, c_m) = 1.$$

Then, for some $1 \le i_0 \le m$ we have $c_{i_0} \not\equiv 0 \pmod{M}$.

One easily verifies that $2^{n-k} = \exp(5m \ln m + O(m))$. Hence, from Lemma 2 we derive that there exists y, $0 \le y \le 2^{n-k}$, such that for $i = 1, \ldots, m$,

$$P(y_1, \ldots, y_{n-k}, s_i) \equiv S(2^k y + p_i) \equiv \begin{cases} 1, & \text{if } i = i_0, \\ 0, & \text{if } i \ne i_0, \end{cases} \pmod{M}$$

where $y = y_1 \ldots y_{n-k}$ is the bit representation of y (with several leading zeroes, if necessary, to make it of length $n-k$). Hence,

$$\sum_{i=1}^{m} c_i P(y_1, \ldots, y_{n-k}, s_i) \equiv c_{i_0} \not\equiv 0 \pmod{M}.$$

From the obtained contradiction we see that $\tau \ge m \ge 2^{(k-1)/2}$. Taking into account that $H(0.1) < 1/2$ and $0.1/\ln 2 \ge 0.14$, from the inequalities (8) and Lemma 1 we obtain the desired result. □

Theorem 2. *Let $M = p^\nu$ be a prime power. Assume that a polynomial*

$$P(X_1, \ldots, X_n) \in \mathbb{Z}_M[X_1, \ldots, X_n]$$

one-sidedly M-represents $S(x)$, that is, it is such that for any x, $1 \le x \le 2^n - 1$,

$$P(x_1, \ldots, x_n) \equiv 0 \pmod{M} \iff S(x) = 0,$$

where $x = x_1 \ldots x_n$ is the bit representation of x. Then, for sufficiently large n, the bounds

$$\deg P \ge 0.14 \ln n \qquad \textit{and} \qquad \operatorname{spr} P \ge \frac{n}{5 \ln n}$$

hold.

Proof. As in the proof of Theorem 1 we obtain that, for some $1 \le i_0 \le m$, and some $u \not\equiv 0 \pmod M$,

$$P(y_1, \ldots, y_{n-k}, s_i) \equiv \begin{cases} u, & \text{if } i = i_0, \\ 0, & \text{if } i \ne i_0, \end{cases} \pmod M.$$

Also $c_{i_0} \not\equiv 0 \pmod p$, and hence, $\gcd(c_{i_0}, M) = 1$. Therefore,

$$\sum_{i=1}^{m} c_i P(y_1, \ldots, y_{n-k}, s_i) \equiv c_{i_0} u \not\equiv 0 \pmod M,$$

and as in the proof of Theorem 1 we obtain the desired result. □

Now we consider deciding if a given n-bit integer is square-free via real polynomials.

Theorem 3. *Let α_0, α_1 be two distinct real numbers, and $n \ge 1$ be an integer. Suppose that a polynomial*

$$f(X_1, \ldots, X_n) \in \mathbb{R}[X_1, \ldots, X_n]$$

is such that for any x, $1 \le x \le 2^n - 1$,

$$\operatorname{sign} f(\alpha_{x_1}, \ldots, \alpha_{x_n}) = S(x),$$

where $x = x_1 \ldots x_n$ is the bit representation of x. Then, for sufficiently large n, the bounds

$$\deg f \ge 0.14 \ln n \qquad \textit{and} \qquad \operatorname{spr} f \ge \frac{n}{5 \ln n}$$

hold.

Proof. We proceed as in the proof of Theorem 2. Assuming that n is large enough, we put

$$m = \left\lceil \frac{n}{5 \ln n} \right\rceil.$$

Let $p_1, \ldots, p_m$ and k be defined as in Lemma 2.

We denote by τ the number of monomials $\mu_j(w)$, $j = 1, \ldots, \tau$, in $w = (w_1, \ldots, w_k)$, such that for every k-dimensional vector

$$w = (w_1, \ldots, w_k) \in \{\alpha_0, \alpha_1\}^k$$

we have a representation of the form

$$f(Y_1, \ldots, Y_{n-k}, w) = \sum_{j=1}^{\tau} \mu_j(w) f_j(Y_1, \ldots, Y_{n-k})$$

with some polynomials $f_j(Y_1, \ldots, Y_{n-k}) \in \mathbb{R}[Y_1, \ldots, Y_{n-k}]$.

Obviously,

$$\tau \le \binom{\deg f + k}{\deg f} \quad \text{and} \quad \tau \le \operatorname{spr} f. \tag{9}$$

As in the proof of Lemma 2, we note that $p_1 < \ldots < p_m < m^2 \le 2^k$. For every $i = 1, \ldots, m$, we add several leading zeroes to the binary representation of p_i to obtain a binary string of length k. In this string we replace 0 by α_0 and 1 by α_1 and denote by $s_i \in \{\alpha_0, \alpha_1\}^k$ this new vector.

If $\tau < m$, then there exist m real coefficients c_i, $i = 1, \ldots, m$, not all equal to zero, at least one of them negative, and such that

$$\sum_{i=1}^{m} c_i \mu_j(s_i) = 0, \qquad j = 1, \ldots, \tau.$$

Therefore, we have the identity:

$$\sum_{i=1}^{m} c_i f(X_1, \ldots, X_{n-k}, s_i) = 0.$$

One can easily verify that

$$2^{n-k} = \exp\left(5m \ln m + O(m)\right).$$

Hence, from Lemma 2 we derive that there exists y, $0 \le y \le 2^{n-k}$, such that:

$$c_i f(\alpha_{y_1}, \ldots, \alpha_{y_{n-k}}, s_i) > 0, \text{ for every } c_i < 0,$$

$$c_i f(\alpha_{y_1}, \ldots, \alpha_{y_{n-k}}, s_i) \ge 0, \text{ for every } c_i \ge 0,$$

where $y = y_1 \ldots y_{n-k}$ is the bit representation of y (with several leading zeroes, if necessary, to make it of length $n - k$). Thus,

$$\sum_{i=1}^{m} c_i f(\alpha_{y_1}, \ldots, \alpha_{y_{n-k}}, s_i) > 0.$$

From the obtained contradiction we see that $\tau \ge m \ge 2^{(k-1)/2}$ and as in the proof of Theorem 1 we obtain the desired result. □

4 Remarks

It is not hard to see that the constants in our estimates can be improved.

On the other hand, we do not know how to obtain more substantial improvements of our lower bounds. In particular, they are exponentially weaker than those which are known for polynomials over $\mathbb{Z}_2$, see [5].

In addition, it would be very interesting to obtain analogues of the results of this paper for other Boolean functions related to various number theoretic problems. For example, for Boolean functions deciding primality or the parity

of the number of prime divisors of x. Unfortunately, even more advanced sieve techniques than those used in Lemma 2 are still not powerful enough to produce such results, even under the assumption of the Extended Riemann Hypothesis.

Finally, it would be very interesting to extend Theorem 2 to arbitrary composite moduli M.

Several more lower bounds on some other important complexity characteristics can be obtained from quite simple considerations.

Let us define the *additive complexity* $\mathrm{C}_{\pm}(f)$ of a polynomial f over reals as the smallest number of '$+$' and '$-$' signs necessary to write down a polynomial [11,16,18,26,27]. Obviously, for any univariate polynomial f

$$\mathrm{C}_{\pm}(f) \le \mathrm{spr}(f) - 1 \le \deg f$$

but neither $\mathrm{spr}(f)$ nor $\deg f$ can be estimated in terms of $\mathrm{C}_{\pm}(f)$. However, it is shown in [18,26,27] that if a non-zero polynomial $f(X) \in \mathbb{R}[X]$ has at least N real zeroes, then

$$\mathrm{C}_{\pm}(f) \ge \left(\frac{1}{5}\log N\right)^{1/2}.$$

The notion of additive complexity is related to the straight-line complexity of f, see [11,16,18,26,27]

Now, let $f(x) \in \mathbb{R}(x)$ be such that

$$\mathrm{sign}\, f(x) = S(x), \qquad 0 \le x \le 2^n - 1.$$

If $4x+1$ is a square-full number and $p > 1$ is a prime number such that $p^2 | (4x+1)$, then $p^2 \equiv 1 \pmod 4$ and $4x + 1 = (4q+1)p^2$ for some positive integer q. For a fixed prime p, there are at most $2^n/p^2 + 1$ integers q that satisfy the above condition. Hence, the number of square-full numbers of the form $4x+1$ is bounded above by

$$\sum_{3 \le p \le (2^n-1)^{1/2}} \left(\frac{2^n}{p^2} + 1\right) \le 2^{n-1}.$$

It follows then, that there is a constant $c > 0$ such that there are at least $c2^n$ square-free numbers of the form $4x+1$ and, thus, $f(4x)f(4x+1) \le 0$ for them. Therefore, $f(x)$ has at least $c2^n$ zeroes. This immediately provides the same bound on the degree of f and the lower bound

$$\mathrm{C}_{\pm}(f) \ge (0.2n)^{1/2} + O(1).$$

Following [22], for a function

$$f : \mathbb{R} \to \{0, 1\}$$

we define the $M_f(n)$*-invariant* as the smallest integer M, such that for any $\lambda < M$ there are two n-bit integers $0 \le x_1 < x_2 \le 2^n - 1$, both divisible

by λ, and such that $f(x_1) \neq f(x_2)$; see also [10,21,22,23] for applications to complexity theory.

It is easy to show that, for any integer λ, there exists $u \leq p^2$ such that $\lambda u + 1$ is square-full, where p is the smallest prime number with $\gcd(\lambda, p) = 1$. Thus $p = O(\log(\lambda + 1))$. It has been shown in [17] that, for any $\varepsilon > 0$, there exists a square-free number of the form $\lambda v + 1$ with $v = O(\lambda^{4/9+\varepsilon})$, where the implied constant depends only on ε.

Therefore, if $f(x) = S(x + 1)$ for $0 \leq x \leq 2^n - 1$, then, for any $\varepsilon > 0$ the bound

$$M_f \geq C(\varepsilon) 2^{9n/13-\varepsilon}$$

holds where $C(\varepsilon) > 0$ depends only on ε.

Acknowledgement

The authors thank Tat-Hung Chan and Alexis Maciel for their interest and several helpful discussions.

References

1. E. Allender, M. Saks and I. E. Shparlinski, 'A lower bound for primality', *J. of Comp. and Syst. Sci.*, **62** (2001), 356–366.
2. D. Barrington, R. Beigel, and S. Rudich, 'Representing Boolean functions as polynomials modulo composite integers', *Comp. Compl.*, **4**, (1994), 367–382.
3. D. Barrington and G. Tardos, 'A lower bound on the MOD 6 degree of the OR function' *Comp. Compl.*, **7** (1998), 99–108.
4. A. Bernasconi, C. Damm and I. E. Shparlinski, 'On the average sensitivity of testing square-free numbers', *Proc. 5th Intern. Computing and Combinatorics Conf.*, Tokyo, 1999, Lect. Notes in Comp. Sci., Springer-Verlag, Berlin, 1999, v.1627, 291–299.
5. A. Bernasconi, C. Damm and I. E. Shparlinski, 'The average sensitivity of square-freeness', *Comp. Compl.* **9** (2000), 39–51.
6. A. Bernasconi, C. Damm and I. E. Shparlinski, 'Circuit and decision tree complexity of some number theoretic problems', *Inform. and Comp.*, **168** (2001), 113–124.
7. A. Bernasconi and I. E. Shparlinski, 'Circuit complexity of testing square-free numbers', *Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1563** (1999), 47–56.
8. J. Bruck, 'Harmonic analysis of polynomial threshold functions', *SIAM J. Discr. Math.*, **3** (1990), 168–177.
9. J. Bruck and R. Smolensky, 'Polynomial threshold functions, $\mathcal{AC}^0$ functions, and spectral norms', *SIAM J. Comp.*, **21** (1992), 33–42.
10. N. H. Bshouty, Y. Mansour, B. Schieber and P. Tiwari, 'Fast exponentiation using the truncation operations', *Comp. Compl.*, **2** (1992), 244–255.
11. P. Bürgisser, M. Clausen and M. A. Shokrollahi, *Algebraic complexity theory*, Springer-Verlag, Berlin, 1996.
12. D. Coppersmith and I. E. Shparlinski, 'On polynomial approximation of the discrete logarithm and the Diffie–Hellman mapping', *J. Cryptology*, **13** (2000), 339–360.

13. H. Davenport, *Multiplicative number theory*, Graduate Texts in Mathematics, 74. Springer-Verlag, New York, 2000.
14. C. Gotsman and N. Linial, 'Spectral properties of threshold functions', *Combinatorica*, **14** (1994), 35–50.
15. F. Green, 'A complex-number Fourier technique for lower bounds on the mod-m degree', *Comp. Compl.*, **9** (2000), 16–38.
16. D. Grigoriev, 'Lower bounds in the algebraic computational complexity', *Zapiski Nauchn. Semin. Leningr. Otdel. Matem. Inst. Acad. Sci. USSR*, **118** (1982), 25–82 (in Russian).
17. D. R. Heath-Brown, 'The least square-free number in an arithmetic progression', *J. Reine Angew. Math.*, **332** (1982), 204–220.
18. A. G. Khovanski, *Fewnomials*, Amer. Math. Soc., Providence, RI, 1997.
19. M. Krause and Pudlák, 'On computing Boolean functions by sparse real polynomials', *Proc. 36th IEEE Symp. on Foundations of Comp. Sci.*, 1995, 682–691.
20. F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, North-Holland, Amsterdam, 1977.
21. Y. Mansour, B. Schieber and P. Tiwari, 'A lower bound for integer greatest common divisor computations', *J. Assoc. Comp. Mach.*, **38** (1991), 453–471.
22. Y. Mansour, B. Schieber and P. Tiwari, 'Lower bounds for computation with the floor operations', *SIAM J. Comp.*, **20** (1991), 315–327.
23. J. Meidânis, 'Lower bounds for arithmetic problems', *Inform. Proc. Letters*, **38** (1991), 83–87.
24. N. Nisan and M. Szegedy, 'On the degree of Boolean functions as real polynomials', *Comp. Compl.*, **4** (1994), 301–313.
25. I. Parberry and P. Yuan Yan, 'Improved upper and lower time bounds for parallel random access machines without simultaneous writes', *SIAM J. Comp.*, **20** (1991), 88–99.
26. J.-J. Risler, 'Khovansky's theorem and complexity theory', *Rocky Mountain J. Math.*, **14** (1984), 851–853.
27. J.-J. Risler, 'Additive complexity of real polynomials', *SIAM J. Comp.*, **14** (1985), 178–183.
28. V. Roychowdhry, K.-Y. Siu and A. Orlitsky, 'Neural models and spectral methods', *Theoretical advances in neural computing and learning*, Kluwer Acad. Publ., Dordrecht, 1994, 3–36.
29. I. E. Shparlinski, *Number theoretic methods in cryptography: Complexity lower bounds*, Birkhäuser, 1999.
30. S.-C. Tsai, 'Lower bounds on representing Boolean functions as polynomials in $\mathbb{Z}_m$', *SIAM J. Discr. Math.*, **9**(1) (1996), 55–62.
31. I. Wegener, *The complexity of Boolean functions*, Wiley-Teubner Series in Comp. Sci., Stuttgart, 1987.
32. A. Woods, 'Subset sum "cubes" and the complexity of prime testing', *Preprint*, 2001, 1–17. Available from `http://www.maths.uwa.edu.au/~woods/alan_research.html`.

Range Allocation for Equivalence Logic

Amir Pnueli[1], Yoav Rodeh[1,2], and Ofer Shtrichman[1,2]

[1] Weizmann Institute of Science, Rehovot, Israel
[2] IBM Haifa Research Laboratory
{amir | yrodeh | ofers}@wisdom.weizmann.ac.il

Abstract. The *range allocation* problem was recently introduced as part of an efficient decision procedure for deciding satisfiability of equivalence logic formulas with or without uninterpreted functions. These type of formulas are mainly used when proving equivalence or refinement between systems (hardware designs, compiler's translation, etc). The problem is to find in polynomial time a small finite domain for each of the variables in an equality formula φ, such that φ is valid if and only if it is valid over this small domain. The heuristic that was presented for finding small domains was *static*, i.e. it finds a small set of integer constants for each variable. In this paper we show new, more flexible range allocation methods. We also show the limitations of these and other related approaches by proving a lower bound on the size of the state space generated by such procedures. To prove this lower bound we reduce the question to a graph theoretic counting question, which we believe to be of independent interest.

1 Introduction

The range allocation problem was introduced in [PRSS98] as part of an efficient decision procedure for equivalence logic formulas with or without uninterpreted functions. These type of formulas are mainly used when proving equivalence or refinement (abstraction) between systems. Deciding satisfiability (and validity) of formulas with uninterpreted functions is of major importance due to their broad use in abstraction. We refer the reader to [BD94] and [PSS98], where these type of formulas are used for proving equivalence between hardware designs (former) and for translation validation, a process in which the correctness of a compiler's translation is proven by checking the equivalence of the source and target codes (latter).

In the past few years several different BDD-base procedures for checking satisfiability of such formulas have been suggested. (in contrast to earlier decision procedures that are based on computing congruence closure [BDL96] in combination with case splitting). Typically the first step of these procedures is the translation of the original formula φ to a function-free formula in equivalence logic ψ such that ψ is satisfiable iff φ is. Then, a procedure for checking satisfiability of E-formulas is used for deciding ψ. This second procedure is the focus of this paper.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 317–333, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Goel et al. suggest in [GSZAS98] to replace all comparisons in ψ with new Boolean variables, and thus create a new Boolean formula ψ'. The BDD of ψ' is calculated ignoring the transitivity constraints of comparisons. They then traverse the BDD, searching for a satisfying assignment that will also satisfy these constraints. Bryant et al. at [BV00] suggested to avoid this potentially exponential traversing algorithm by explicitly computing a small set of constraints that are sufficient for preserving the transitivity constraints of equality. By checking ψ' conjuncted with these constraints using a regular BDD package they were able to verify larger designs.

The method which we will present here, extends the method first presented in [PRSS98], where ψ's satisfiability is decided by allocating a small domain for each variable, such that ψ is satisfiable if and only if it is satisfiable over this small domain. To find this domain, the equalities in the formula are represented as a graph, where the nodes are the variables and the edges are the equalities and disequalities (*disequality* standing for $\neq$) in ψ. Given this graph, a heuristic called *range allocation* is used in order to compute a small set of values for each variable. To complete the process, a standard BDD based tool is used to check satisfiability of the formula over the computed domain. In [RS01] we elaborate on this by generating a smaller graph than [PRSS98]. This is achieved by examining the original formula with uninterpreted function φ, instead of its translated version ψ.

In this paper, we extend the second part of [PRSS98], by suggesting a more general method of allocating finite domains to variables. Using the information in the graph generated by [PRSS98] or [RS01], we suggest and analyze different procedures for generating a small state space that is adequate for checking ψ. One of our main results is a general lower bound on the size of the state space generated by any method using only the information in this graph. The suggests the need for a more in depth investigation of the formula at hand, rather than only examining which atomic equalities appear in it.

2 Equivalence Logic Formulas

An equivalence logic formula (called an E-formula) has the following syntax:

$$\langle Formula\rangle \longleftarrow \langle Term\rangle = \langle Term\rangle \mid \neg\langle Formula\rangle \mid \langle Formula\rangle \vee \langle Formula\rangle$$

$$\langle Term\rangle \longleftarrow \langle Variable\rangle \mid \mathbf{ITE}(\langle Formula\rangle, \langle Term\rangle, \langle Term\rangle)$$

$\mathbf{ITE}(f, t, e)$ stands for **if** f **then** t **else** e. The E-formula φ is said to be satisfiable if there is some assignment of values to φ's variables that satisfies φ. Therefore, an E-formula φ with variables V is a function $\varphi : \mathbb{N}^V \to \{0, 1\}$. However, not all such functions can be realized as E-formulas, for example $\varphi(a, b) = a > b$. Therefore, we will try to make a more accurate definition.

Definition 1. (partition): *Given a set V, we say that $\alpha = \{\alpha_1, \ldots, \alpha_k\}$ is a* partition *of V if:*

1. $V = \alpha_1 \cup \alpha_2 \cup \ldots \cup \alpha_k$
2. *for all* $i \neq j$ *we have that* $\alpha_i \cap \alpha_j = \emptyset$

Given a partition $\alpha = \{\alpha_1, \ldots, \alpha_k\}$ we denote $u \sim_\alpha v$ if there is $\alpha_i \in \alpha$ such that $u, v \in \alpha_i$. In other words, a partition α gives us an equivalence relation $\sim_\alpha$. In fact, there is a one-to-one correspondence between the set of equivalence relations on V and its set of partitions.

Given an assignment to the variables of V, $a : V \to \mathbb{N}$, we denote by $partition(a)$ the partition of V that satisfies $a(v) = a(u) \iff v \sim_{partition(a)} u$. The following is immediate from the fact that E-formulas only query comparisons between their input variables:

Claim. For E-formula φ, if assignments a, b satisfy $partition(a) = partition(b)$ then $\varphi(a) = \varphi(b)$.

This means that E-formula φ on variables V can be viewed as a function of the partitions of V instead of as a function of assignments to V. Denote by $P(V)$ the set of partitions of the set V. φ can therefore be viewed as a function $\varphi : P(V) \to \{0, 1\}$. In fact, any function from $P(V)$ to $\{0, 1\}$ can be realized as an E-formula with variable set V.

It is easy to verify that letting each variable in an equivalence formula range over $\{1, \ldots, |V|\}$ suffices for checking the formulas satisfiability. This shows that deciding satisfiability of equivalence formulas is in NP, and therefore is clearly NP-complete (proved via a trivial reduction from satisfiability of boolean formulas).

3 Equivalence Graphs

Definition 2. (E-graph): *An* E-graph $\mathcal{G}$ *is a triple* $\mathcal{G} = \langle V, EQ, DQ \rangle$ *where* V *is the set of vertices, and* EQ *(Equality edges) and* DQ *(disequality edges) are sets of unordered pairs from* V*.*

We will use $\mathcal{G}$ and $\mathcal{H}$ to denote E-graphs, and G and H for standard graphs. For E-graph $\mathcal{G} = \langle V, EQ, DQ \rangle$ we denote $V(\mathcal{G}) = V$, $EQ(\mathcal{G}) = EQ$ and $DQ(\mathcal{G}) = DQ$. We denote by $\mathcal{G}_=$ the graph on vertices $V(\mathcal{G})$ and edges $EQ(\mathcal{G})$. We use $\leq$ to denote the subgraph relation: $\mathcal{H} \leq \mathcal{G}$ if $EQ(\mathcal{H}) \subseteq EQ(\mathcal{G})$ and $DQ(\mathcal{H}) \subseteq DQ(\mathcal{G})$.

We say that a partition α satisfies equality edge (u, v) if $u \sim_\alpha v$, and that it satisfies inequality edge (u, v) if $u \not\sim_\alpha v$. We say that a partition α satisfies E-graph $\mathcal{G}$ (denoted $\alpha \models \mathcal{G}$) if it satisfies all of $\mathcal{G}$'s edges. An E-graph is said to be satisfiable if there exists a partition that satisfies it.

Lemma 1. *An E-graph* $\mathcal{G}$ *is satisfiable iff for every* $(u, v) \in DQ(\mathcal{G})$*,* u *and* v *are not connected in* $\mathcal{G}_=$*.*

The algorithms of [PRSS98,RS01] construct for a given E-formula φ, an E-graph $\mathcal{G}$, satisfying the following property:

Definition 3. (adequacy of E-graphs for E-formulas): *The E-graph* $\mathcal{G}$ *is* adequate *for the E-formula* φ *if either* φ *is not satisfiable, or there exists a satisfiable* $\mathcal{H} \leq \mathcal{G}$ *such that any partition* $\alpha \models \mathcal{H}$ *satisfies* $\varphi(\alpha) = 1$*.*

Hence, if we want to check whether φ is satisfiable, we only need to check φ on a relatively small set of partitions:

Definition 4. (adequacy of partition sets for E-graphs): *For an E-graph $\mathcal{G}$ and a set of partitions $R \subseteq P(V(\mathcal{G}))$, we say that R is* adequate *for $\mathcal{G}$ iff for every satisfiable $\mathcal{H} \leq \mathcal{G}$, there is $\alpha \in R$ such that $\alpha \models \mathcal{H}$.*

This leads to the following claim:

Claim. If the partition set R is adequate for the E-graph $\mathcal{G}$ and $\mathcal{G}$ is adequate for the E-formula φ, then φ is satisfiable iff there is some $\alpha \in R$ such that $\varphi(\alpha) = 1$.

To use this claim, we need to devise a procedure that, given an E-graph $\mathcal{G}$, will find a small partition set R which is adequate for $\mathcal{G}$.

4 Static Range Allocation

The method of [PRSS98] generates for an E-formula φ, an E-graph $\mathcal{G}$ which is adequate for φ, and then uses it to find for each variable v of φ, a small range of natural numbers $D(v)$. It is then proved that φ is satisfiable iff it is satisfiable where each variable v is allowed to range only over $D(v)$ (as opposed to $\mathbb{N}$).

Therefore, we check φ over the assignment set $A_D = \{a \mid \forall v, a(v) \in D(v)\}$. The corresponding partition set is $P_D = \{partition(a) \mid a \in A_D\}$, and the construction of [PRSS98] guarantees that P_D is adequate for $\mathcal{G}$. In general, we will say that assignment set A is adequate for an E-graph $\mathcal{G}$ if its corresponding partition set is adequate for $\mathcal{G}$.

All methods proposed so far (including this paper), generate a new Boolean formula from the original E-formula φ. For example, the static range allocation method replaces every variable of φ by a variable ranging over a finite domain. This resulting formula is then translated to a Boolean formula using standard methods.

We will therefore measure the complexity of our proposed methods in terms of the number of Boolean variables in the newly generated Boolean formula (or equivalently, the size of the state space checked, which is 2 to the power of the number of Boolean variables). For example, in static range allocation, the state space size will be $\prod_v |D(v)|$, i.e., the number of boolean variables needed is $\sum_v \log(|D(v)|)$. This complexity measure is not always related to the true complexity of checking the resulting formula, but it is usually a good indicator for it.

5 Dynamic Range Allocation

We propose the new method of *dynamic range allocation* which improves upon the static range allocation method. Both experimental (Section 10) and theoretical (Section 9) results show the advantages of dynamic over static range allocation.

Definition 5. (dynamic assignment): *The mapping* $x : V \to \mathbb{N} \cup V$ *is a* dynamic assignment *if it is acyclic; i.e., there exists an ordering of* V, $v_1, \ldots, v_n$ *such that,* $x(v_i) = v_j$ *implies* $j > i$.

Definition 6. (induced assignment): *For the dynamic assignment* x*, we define the* induced *(static) assignment* $\overline{x} : V \to \mathbb{N}$*:*

1. *If* $x(v) \in \mathbb{N}$ *then* $\overline{x}(v) = x(v)$
2. *If* $x(v) = u \in V$ *then* $\overline{x}(v) = \overline{x}(u)$.

Note that this is a recursive definition, and that it is well defined since we required that dynamic assignments be acyclic.

Example 1. The dynamic assignment: $x(v_1) = v_2, x(v_2) = v_3$ and $x(v_3) = 2$, induces the static assignment $\overline{x}$: $\overline{x}(v_1) = 2, \overline{x}(v_2) = 2, \overline{x}(v_3) = 2$.

For E-graph $\mathcal{G}$ and dynamic assignment x we denote $x \models \mathcal{G}$ if $\overline{x} \models \mathcal{G}$.

Definition 7. (dynamic range): *A* dynamic range *for vertex set* V *is a function* $D : V \to 2^{\mathbb{N} \cup V}$ *that is acyclic; i.e. there exists an ordering of* V, $v_1, \ldots, v_k$ *such that,* $v_j \in x(v_i)$ *implies* $j > i$.

A dynamic range D gives rise to $X_D = \{x \mid \forall v \in V, x(v) \in D(v)\}$, a set of dynamic assignments, which in turn induces a set of static assignments $\overline{X_D} = \{\overline{x} \mid x \in X_D\}$. We will therefore say that D is adequate for the E-graph $\mathcal{G}$ if the above assignment set $\overline{X_D}$ is adequate for $\mathcal{G}$.

One advantage of dynamic range allocation is that given an E-formula φ, and an adequate dynamic range D for φ, we can efficiently generate a Boolean formula that is satisfiable iff φ is, with little increase in the size of the formula.

W.l.o.g., assume that the variable ordering satisfying Definition 7 is $v_1, \ldots, v_n$, and therefore, fore every i, $D(v_i) \subseteq \mathbb{N} \cup \{v_i + 1, \ldots, v_n\}$. We encode the range $D(v_i)$ by a variable c_i whose domain is:

$$D(c_i) = (\mathbb{N} \cap D(v_i)) \cup \{-j \mid v_j \in D(v_i)\}$$

That is, we include in $D(c_i)$ all the integers which are in $D(v_i)$ and, for each $v_j \in D(v_i)$, we include $-j$ in $D(c_i)$.

Then, we can derive a formula χ which is satisfiability equivalent to φ but depends only on the variables $c_1, \ldots, c_n$. If these variables are then finite, i.e., all $D(v_i)$ are finite, then χ can be easily translated to a Boolean formula. We let

$$\chi_1 = \textbf{ let } (v_1 = \textbf{ if } (c_1 \geq 0) \textbf{ then } c_1 \textbf{ else } v_{-c_1}) \textbf{ in } \varphi$$

We then derive $\chi_2, \chi_3, \ldots, \chi_n$ successively, where for each $i > 1$:

$$\chi_i = \textbf{ let } (v_i = \textbf{ if } (c_i \geq 0) \textbf{ then } c_i \textbf{ else } v_{-c_i}) \textbf{ in } \chi_{i-1}$$

Finally, we let $\chi = \chi_n$. Note that since the dynamic range is acyclic, the resulting formula (with term-sharing) is also acyclic.

Claim. If D is adequate for $\mathcal{G}$ which is adequate for φ, the resulting formula χ is satisfiable iff φ is.

We now wish to construct a procedure that, given an E-graph $\mathcal{G}$, will construct a small adequate dynamic range for it. Notice that like in static ranges, the size of the state space when using a dynamic range D is $\prod_v |D(v)|$.

We will use the following notation for an E-graph $\mathcal{G}$ and a vertex $v \in V(\mathcal{G})$: $\Gamma_{EQ}(v) = \{u \mid (u,v) \in EQ(\mathcal{G})\}$, and $\Gamma_{DQ}(v) = \{u \mid (u,v) \in DQ(\mathcal{G})\}$.

We now define the E-graph $\mathcal{G}[v]$, which results from removing the vertex v from $\mathcal{G}$ and transforming v's equality and disequality constraints to constraints on its neighboring vertices:

1. The vertex set is $V(\mathcal{G}) \setminus \{v\}$.
2. Initially, $\mathcal{G}[v]$'s edges are all the edges of $\mathcal{G}$ which are not adjacent to v.
3. For $u_1 \neq u_2$ and $u_1, u_2 \in \Gamma_{EQ}(v)$, add an equality edge (u_1, u_2).
4. For $u_1 \neq u_2$, $u_1 \in \Gamma_{DQ}(v)$ and $u_2 \in \Gamma_{EQ}(v)$. add a disequality edge (u_1, u_2).

Example 2. In Figure 1, $\mathcal{G}_1 = \mathcal{G}_0[a]$, $\mathcal{G}_2 = \mathcal{G}_1[b]$, $\mathcal{G}_3 = \mathcal{G}_2[c]$, etc.

The following theorem is our building block for the construction of procedures that calculate an adequate dynamic range for a given E-graph $\mathcal{G}$ (see Appendix A.1 for its proof):

Theorem 1. *For E-graph $\mathcal{G}$ and $u \in V(\mathcal{G})$, if the dynamic range D is adequate for $\mathcal{G}[u]$, then D' is adequate for $\mathcal{G}$, where D' is defined as follows:*

1. *$D'(v) = D(v)$ for every $v \neq u$.*
2. *If $\Gamma_{DQ}(u) = \emptyset$ and $\Gamma_{EQ}(u) \neq \emptyset$ then $D'(u) = \Gamma_{EQ}(u)$. Otherwise, $D'(u) = \Gamma_{EQ}(u) \cup \{\mathbf{unique}\}$.*

Where $\mathbf{unique} \in \mathbb{N}$ *and* $\mathbf{unique} \notin \cup_v D(v)$.

Based on Theorem 1, the following procedure finds an adequate dynamic assignment set for a given E-graph $\mathcal{G}$:

1. Set $counter \leftarrow 0$.
2. Pick some vertex v of $V(\mathcal{G})$.
3. Set:

$$D(v) = \begin{cases} \{counter\} & \Gamma_{EQ}(v) = \emptyset \\ \Gamma_{EQ}(v) \cup \{counter\} & \Gamma_{EQ}(v) \neq \emptyset \text{ and } \Gamma_{DQ}(v) \neq \emptyset \\ \Gamma_{EQ}(v) & \Gamma_{EQ}(v) \neq \emptyset \text{ and } \Gamma_{DQ}(v) = \emptyset \end{cases}$$

4. Set $\mathcal{G} = \mathcal{G}[v]$.
5. Set $counter \leftarrow counter + 1$.
6. If $V(\mathcal{G}) \neq \emptyset$ return to Step 2.

Notice that *counter* is only used to generate unique numbers.

Example 3. Using this procedure we can generate an adequate dynamic range for the E-graph $\mathcal{G}_0$ of Figure 1:

1. Set $D(a) = \{c\}$ and calculate $\mathcal{G}_1 = \mathcal{G}_0[a]$.
2. Set $D(b) = \{0, d, e\}$, and $\mathcal{G}_2 = \mathcal{G}_1[b]$.
3. Set $D(c) = \{1, d, e\}$, and $\mathcal{G}_3 = \mathcal{G}_1[c]$.
4. Set $D(d) = \{2, e\}$, and $\mathcal{G}_4 = \mathcal{G}_3[d]$.

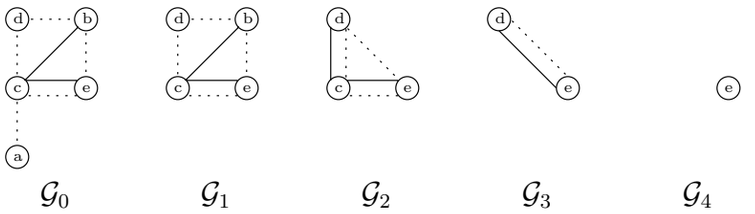


Fig. 1. Dynamic range allocation is based on an incremental process in which vertices are removed one by one, and their constraints are reallocated to their neighbors. The dashed lines in graphs $\mathcal{G}_0, \ldots, \mathcal{G}_4$ represent equality edges while solid lines represent disequality edges.

5. Set $D(e) = \{3\}$.

The resulting state space in our example was of size 18. If we would have taken a different order on the vertices when using our procedure: a, d, b, c, e, the resulting state space would be of size 12. In our implementation we use a simple greedy heuristic that in Step 2 chooses the vertex which will be allocated the smallest domain in Step 3. This heuristic generates a state space of 12 on the above example. In Section 10 we compare this procedure with the static range allocation procedure of [PRSS98].

The sequence of E-graphs generated by our procedure is almost the same as that generated by the procedure suggested by [BV00], except they don't distinguish between equality and disequality edges, and therefore the vertices in our graphs should generally have a smaller degree. In their procedure, when a vertex is removed from the graph, the number of Boolean variables added to the formula is equal to its degree, and therefore the number of Boolean variables appearing in their resulting formula is $\sum_v degree(v)$, where $degree(v)$ is the degree of v in the graph at the time of its removal. In contrast, in our procedure, when a vertex v_i is removed, a variable c_i ranging over $degree(v) + 1$ values is added to the formula, and so we get $\sum_v \log(degree(v) + 1)$ Boolean variables. Therefore, we need much less Boolean variables than their method. However, we note that the Boolean formula they generate is very different than ours, and in spite of the increased number of boolean variables, may actually be easier to check.

6 One-Orientable Assignment Sets

In this section we present an alternative method for finding a partition set that is adequate for a given E-graph. This method generates a partition set of a different kind, with a more complex representation. Although the resulting Boolean formula has a relatively small number of Boolean variables in it, this method is not practical since this Boolean formula is much larger and more complex than the original one. This renders this method impractical, yet still interesting since

we can show that on a large set of E-graphs the number of Boolean variables is slightly more than twice the minimal number needed.

Definition 8. (partition of a graph): *A graph G defines a partition:*

$$\alpha_G = \{W \subseteq V(G) \mid W \text{ is a connected component of } G\}$$

Definition 9. (one-orientable): *A graph is* one-orientable *if its edges can be directed in such a way that the out-degree of every vertex* ≤ 1.

Definition 10. (one-orientable partition set): *The* one-orientable partition set *of an E-graph $\mathcal{G}$ is $One(\mathcal{G}) = \{\alpha_H \mid$ graph H is one-orientable and $H \leq \mathcal{G}_= \}$*

Proposition 1. *$One(\mathcal{G})$ is adequate for $\mathcal{G}$.*

Example 4. Figure 2 presents the graph $\mathcal{G}_=$ of some E-graph $\mathcal{G}$. The sub-graphs $1, \ldots, 5$ describe some of its one-orientable sub-graphs, and therefore their corresponding partitions are in $One(\mathcal{G})$; i.e., $\{\{a, b, c, d\}\} \in One(\mathcal{G})$ because of sub-graph 1, $\{\{a\}, \{b\}, \{c\}, \{d\}\}$ because of 2, and $\{\{a, b\}, \{c, d\}\}$ because of 5. Note that the only sub-graph of this graph that is not one-orientable is the graph itself.

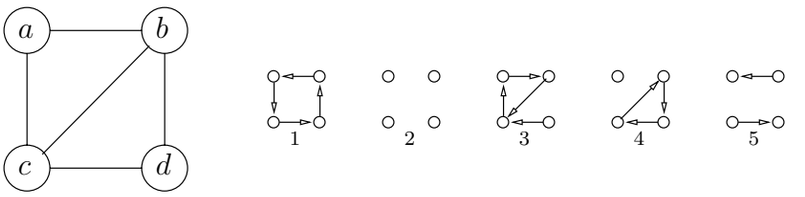


Fig. 2. The small graphs are one-orientable sub-graphs of the graph on the left. The direction on the edges was added for demonstrating the fact that the sub-graphs are one-orientable.

As a result, if we are given an E-formula φ together with an adequate E-graph $\mathcal{G}$ for it, we can check if φ is satisfiable by checking if there is some partition $\alpha \in One(\mathcal{G})$ such that $\varphi(\alpha) = 1$. This is implemented as follows:

Represent φ as a Boolean formula with atoms $(u = v)$. This can be done by flattening all the **ITE** terms in the formula. This flattening procedure increases the size of the formula only polynomially. Now construct the following formula C:

1. For each $v \in V(\mathcal{G})$, C has an input variable I_v ranging over $\{u \mid (u, v) \in EQ(\mathcal{G})\} \cup \{\star\}$.
2. For each $u, v \in V(\mathcal{G})$, C contains an internal variable $e_{(u,v)} := (I_u = v) \vee (I_v = u)$.

3. C contains a circuit for calculating the transitive closure of a graph with vertices $V(\mathcal{G})$. for every $u, v \in V(\mathcal{G})$, it has input variable $e_{(u,v)}$, and output variable $t_{(u,v)}$. There are known constructions of this circuit that are of polynomial size (e.g., using successive $\log(V(\mathcal{G}))$ boolean matrix multiplications).
4. Replace every atom $(u = v)$ in φ by $t_{(u,v)}$.

Proposition 2. *C is satisfiable iff φ is satisfiable.*

The general idea of the construction (see Appendix A.2 for the proof) is that the variables $e_{u,v}$ represent a one-orientable sub-graph of $\mathcal{G}_=$ that results from undirecting all the edges (u, I_u). Then $t_{u,v}$ represents the partition resulting from this sub-graph, that is used as input to φ.

The size of the resulting state space is $\prod_v |I_v| = \prod_{v \in V}(degree(v)+1)$, where $degree(v)$ is the degree of v in $\mathcal{G}_=$.

7 Lower Bound on The Size of Partition Sets

In Section 6, we constructed an adequate partition set (the one-orientable partition set) for a given E-graph $\mathcal{G}$. This set was of size at most $\prod_{v \in V}(degree(v)+1)$, where $degree(v)$ is the degree of v in $\mathcal{G}_=$. In this section we show every partition set that is adequate for $\mathcal{G}$ is at least of size:

$$\sqrt{\prod_{v \in V}(\frac{1}{2}degree'(v) + 1)}$$

where $degree'(v)$ is the degree of v in $\langle V(\mathcal{G}), EQ(\mathcal{G}) \cap DQ(\mathcal{G})\rangle$. Therefore on any E-graph where $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$, $One(\mathcal{G})$ is close to optimal in terms of the number of Boolean variables that is needed to represent such a state space. We get that any adequate assignment set will need at least $\frac{1}{2}\sum(\log(degree(v)+2)-1)$ Boolean variables, and that for representing $One(\mathcal{G})$ we need only slightly more than twice this number: $\sum \log(degree(v)+1)$.

It is important to notice that a lower bound on the size of an adequate partition set for the E-graph $\mathcal{G}$ is in fact a lower bound on the size of the state space generated by any method that uses only the information in $\mathcal{G}$. This means that this lower bound applies to any method that examines only the set of atomic equalities (and their polarity) that appear in the E-formula. To break this lower bound barrier, a more careful analysis of the formula will be needed.

We start with the following definition:

Definition 11. (maximal satisfiable sub-graph): *An E-graph $\mathcal{H}$ is a* maximal satisfiable sub-graph *of $\mathcal{G}$ (denoted $\mathcal{H} \lll \mathcal{G}$), if it is a satisfiable sub-graph of $\mathcal{G}$, and there is no $\mathcal{H}_1$ that is a satisfiable sub-graph of $\mathcal{G}$ such that $\mathcal{H}$ is a proper subgraph of $\mathcal{H}_1$.*

Lemma 2. *If $\mathcal{H}_1 \lll \mathcal{G}$, $\mathcal{H}_2 \lll \mathcal{G}$ and $\mathcal{H}_1 \neq \mathcal{H}_2$ then there is no partition α such that $\alpha \models \mathcal{H}_1$ and $\alpha \models \mathcal{H}_2$.*

Proof. Assume to the contrary: $\alpha \models \mathcal{H}_1$ and $\alpha \models \mathcal{H}_2$. Define E-graph $\mathcal{H} = \langle V(\mathcal{G}), EQ(\mathcal{H}_1) \cup EQ(\mathcal{H}_2), DQ(\mathcal{H}_1) \cup DQ(\mathcal{H}_2) \rangle$. Clearly, $\alpha \models \mathcal{H}$. Also, $\mathcal{H} \leq G$, and thereby $\mathcal{H} \lll \mathcal{G}$. Since $\mathcal{H}_1, \mathcal{H}_2 \leq \mathcal{H}$, $\mathcal{H} = \mathcal{H}_1 = \mathcal{H}_2$. □

Lemma 2 directly implies:

Corollary 1. *If partition set R is adequate for the E-graph $\mathcal{G}$, then $|R| \geq |\{\mathcal{H} \mid \mathcal{H} \lll \mathcal{G}\}|$.*

Consider an E-graph $\mathcal{G}$ where $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$. For this type of E-graphs we can bound their set of maximal satisfiable sub-graphs. We say a partition α is connected in a graph G if every set $\alpha_i \in \alpha$ is connected in G restricted to the vertices of α_i. We denote by $CP(G)$ the set of connected partitions of G.

Example 5. In the graph of Figure 2, $\{\{a,b\},\{c\},\{d\}\}$ and $\{\{a,c,d\},\{b\}\}$ are connected partitions, yet $\{\{c,b\},\{a,d\}\}$ is not.

Proposition 3. *If $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$, then $|CP(\mathcal{G}_=)| \leq |\{\mathcal{H} \mid \mathcal{H} \lll \mathcal{G}\}|$*

Proof. We define the following mapping:

$$\psi : \{\mathcal{H} \mid \mathcal{H} \lll \mathcal{G}\} \to CP(\mathcal{G}_=)$$

Where $\psi(\mathcal{H}) = \alpha_{\mathcal{H}_=}$ (see Definition 8).

Clearly, for every $\mathcal{H}$, $\psi(\mathcal{H}) \in CP(\mathcal{G}_=)$ since for every $\alpha_i \in \alpha_{\mathcal{H}_=}$, α_i is a connected component of $\mathcal{H}_= \leq \mathcal{G}_=$, and so α_i is connected in $\mathcal{G}_=$.

To prove the proposition we show that ψ is onto. For $\alpha \in CP(\mathcal{G}_=)$, define $\mathcal{H} \leq \mathcal{G}$ to be:

1. If $u \sim_\alpha v$ and $(u,v) \in EQ(\mathcal{G})$ then $(u,v) \in EQ(\mathcal{H})$.
2. If $u \not\sim_\alpha v$ and $(u,v) \in DQ(\mathcal{G})$ then $(u,v) \in DQ(\mathcal{H})$.

We claim that $\mathcal{H}$ is maximal and that $\psi(\mathcal{H}) = \alpha$ (clearly $\mathcal{H}$ is satisfiable). To show that $\psi(\mathcal{H}) = \alpha$, we need to show that the connected components of $\mathcal{H}_=$ coincide with the sets of α:

1. Each connected component of $\mathcal{H}_=$ is a subset of some $\alpha_i \in \alpha$, since every edge $(u,v) \in \mathcal{H}_=$ satisfies $u \sim_\alpha v$.
2. Each α_i is a subset of some connected component of $\mathcal{H}_=$, since all edges of $\mathcal{G}_=$ between vertices of α_i are in $\mathcal{H}_=$, and α_i is connected in $\mathcal{G}_=$.

We now show that $\mathcal{H}$ is maximal. We have to handle two cases:

1. $(u,v) \in EQ(\mathcal{G})$, $(u,v) \notin EQ(\mathcal{H})$. This means that $u \not\sim_\alpha v$, and therefore, since $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$, then $(u,v) \in DQ(\mathcal{H})$, but then if we add (u,v) to $EQ(\mathcal{H})$ it makes $\mathcal{H}$ unsatisfiable.
2. $(u,v) \in DQ(\mathcal{G})$, $(u,v) \notin DQ(\mathcal{H})$. This means that $u \sim_\alpha v$, implying there exists $\alpha_i \in \alpha$ such that $u, v \in \alpha_i$. α_i is connected in $\mathcal{G}_=$, and therefore in $\mathcal{H}_=$. Now, using Lemma 1, we see that adding (u,v) to $DQ(\mathcal{H})$ will make $\mathcal{H}$ unsatisfiable.

□

Corollary 2. *For an E-graph $\mathcal{G}$ such that $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$, every partition set R that is adequate for $\mathcal{G}$ satisfies $|R| \geq |CP(\mathcal{G}_=)|$.*

Theorem 2. *For every graph G, $|CP(G)| \geq \sqrt{\prod_{v \in V(G)} (\frac{1}{2} degree(v) + 1)}$*

We believe this theorem (and its proof) to be of independent interest, as it addresses a natural (yet nontrivial) combinatorial counting question: the number of connected partitions of a given graph.

In fact, using a construction similar to that of one-orientable sets, it is easy to show that for every graph G, $|CP(G)| \leq \prod_{v \in V(G)} (degree(v) + 1)$. Together with Theorem 2 we have a good approximation of $|CP(G)|$.

Combining Theorem 2 with Corollary 2 we get:

Corollary 3. *For an E-graph $\mathcal{G}$ such that $EQ(\mathcal{G}) \subseteq DQ(\mathcal{G})$, every partition set R that is adequate for $\mathcal{G}$ satisfies*

$$|R| \geq \sqrt{\prod_{v \in V(G)} (\frac{1}{2} degree(v) + 1)}$$

Where $degree(v)$ is the degree of v in $\mathcal{G}_=$.

Claim. For any two E-graphs $\mathcal{G}_1 \leq \mathcal{G}_2$, if partition set R is adequate for $\mathcal{G}_2$ then it is also adequate for $\mathcal{G}_1$.

Using this claim with Corollary 3:

Corollary 4. *For an E-graph $\mathcal{G}$, every partition set R that is adequate for $\mathcal{G}$ satisfies*

$$|R| \geq \sqrt{\prod_{v \in V(G)} (\frac{1}{2} degree'(v) + 1)}$$

Where $degree'(v)$ is the degree of v in $\langle V(\mathcal{G}), EQ(\mathcal{G}) \cap DQ(\mathcal{G}) \rangle$.

8 Proof of Theorem 2

In order to prove Theorem 2 we need the following lemmas:

Lemma 3. *For a graph G and two non-intersecting sets $S, T \subseteq V(G)$ such that $S \cup T = V(G)$*

$$|CP(G)| \geq \prod_{v \in S} (degree_T(v) + 1)$$

Where $degree_T(v) = |\{u \in T \mid (u, v) \in E\}|$.

Proof. Consider the following procedure that constructs a partition α:

1. First start with $\alpha = \{\{v\} \mid v \in T\}$ (Note that α is not a partition yet).
2. For all vertices $v \in S$ do one of the two:

(a) Take one vertex $u \in T$ such that $(u, v) \in E$, and add v to u's set in α.
(b) Add the set $\{v\}$ to α.

Clearly, the final α is a partition, and is connected, since we add a vertex v to a set only if there is a vertex u in that set such that $(u, v) \in E$.

For each vertex $v \in S$, we have to choose between $degree_T(v) + 1$ choices in the procedure. So, if we show that different choices for the vertices always lead to different partitions, then we constructed a set of $\prod_{v \in S}(degree_T(v) + 1)$ different connected partitions of G.

First note that the partition α constructed has the following property: Every $\alpha_i \in \alpha$ satisfies one of:

1. $\alpha_i \cap T = \emptyset$ and $|\alpha_i| = 1$
2. $|\alpha_i \cap T| = 1$.

From this we see that two different runs lead to different partitions. If a vertex v chooses to join some $u \in T$, it will not be in the same set with any other $u' \in T$. So different choices here lead to different partitions. Also, if a vertex chooses not to join any $u \in T$, then it will be a singular set in α, and this cannot happen if it chooses to join some vertex of T. □

Lemma 4. *For a graph G, there are non-intersecting sets S_0, S_1, such that $S_0 \cup S_1 = V(G)$, and for every $v \in S_i$, $degree_{S_{(1-i)}}(v) \geq \frac{1}{2} degree(v)$.*

Proof. Start with two arbitrary sets S_0 and S_1. While possible, pick some $v \in S_i$ such that $degree_{S_{(1-i)}}(v) < \frac{1}{2} degree(v)$. Take v from S_i and put it in S_{1-i}. If there are no such vertices left, then S_0 and S_1 satisfy the lemma.

We claim that the procedure will end. This is because each such move increases the following function:

$$Cut(S_0, S_1) = |\{(u, v) \in E(G) \mid u \in S_0, v \in S_1\}|$$

This is because:

$$Cut(S_0 \cup \{v\}, S_1 \setminus \{v\}) = Cut(S_0, S_1) + degree_{S_1}(v) - degree_{S_0}(v)$$

By the way we pick our vertices, we see that this function increases for each move we make. Since $Cut(S_0, S_1) < |E(G)|$, the procedure will halt after at most $|E(G)|$ moves. □

We comment that the proof of Lemma 4 is in fact a schoolbook construction of a maximum cut in a graph.

Proof. (of Theorem 2): Using Lemma 4 we get two sets non-intersecting sets S_0 and S_1. Now we use Lemma 3, setting $S = S_0$, and $T = S_1$. We get:

$$|CP(G)| \geq \prod_{v \in S_0} (degree_{S_1}(v) + 1) \geq \prod_{v \in S_0} (\frac{1}{2} degree(v) + 1)$$

We use Lemma 3 again, setting $S = S_1$ and $T = S_0$. We get:

$$|CP(G)| \geq \prod_{v \in S_1} (\frac{1}{2} degree(v) + 1)$$

Combining:

$$|CP(G)|^2 \geq \prod_{v \in S_0} (\frac{1}{2} degree(v) + 1) \cdot \prod_{v \in S_1} (\frac{1}{2} degree(v) + 1) = \prod_{v \in V} (\frac{1}{2} degree(v) + 1)$$

Proving the theorem. □

9 Static vs. Dynamic Ranges

It is difficult to estimate the advantages of using dynamic ranges. We cannot show any relation to the minimal partition set possible (as we did for one-orientable ranges), but since dynamic ranges generalize static ranges, they are at least as good. There are E-graphs where dynamic ranges seem to not give any advantage over static ones, but we will show an example where using a dynamic range gives far better results.

Consider the following E-graph $\mathcal{G}$:

1. $V = \{v_i \mid i \in \{1, \ldots, n\}\} \cup \left\{u_i^j \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, d\}\right\}$
2. $DQ = \{(v_i, v_j) \mid i, j \in \{1, \ldots, n\}\}$
3. $EQ = \{(v_i, v_j) \mid i, j \in \{1, \ldots, n\}\} \cup \left\{(u_i^j, v_i) \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, d\}\right\}$

In other words, $\mathcal{G}$ is constructed of a clique of equality and disequality edges: $v_1, \ldots, v_n$, where each v_i has d equality leaves connected to it $u_i^1, \ldots, u_i^d$.

Assume D is some static range of minimal size that is adequate for $\mathcal{G}$.

Claim. for all i and j, $D(v_i) \subseteq D(u_i^j)$

Proof. Assume there is some $l \in D(v_i) \setminus D(u_i^j)$. Define range D' to be the same as D, except $D'(v_i) = D(v_i) \setminus \{l\}$. We claim that D' is adequate for $\mathcal{G}$ contradicting the fact that D is minimal.

For a satisfiable $\mathcal{H} \leq \mathcal{G}$, there exists an assignment $a \in D$ s.t. $a \models \mathcal{H}$. We find $a' \in D'$ s.t. $a' \models \mathcal{H}$. We split to two cases:

1. If $(v_i, u_i^j) \in EQ(\mathcal{H})$ then $a(v_i) = a(u_i^j)$, and therefore, since $a(u_i^j) \neq l$, $a(v_i) \neq l$, and then we define $a' = a$, since $a \in D'$.
2. If $(v_i, u_i^j) \notin EQ(\mathcal{H})$ then adding this edge to $\mathcal{H}$ will leave it satisfiable (this is because of $\mathcal{G}$'s structure). Therefore, there is some $a \in D$ that satisfies this new $\mathcal{H}$. This a will also satisfy the original $\mathcal{H}$. Using the same argument as in (1) we see that $a \in D'$.

□

We therefore have that for all i and j, $|D(u_i^j)| \geq |D(v_i)|$, and therefore:

$$\prod_{i,j} |D(u_j^i)| \geq \left(\prod_i |D(v_i)| \right)^d$$

Considering the restriction of $\mathcal{G}$ to $\{v_1, \ldots, v_n\}$, and using corollary 4, we have:

$$\prod_i |D(v_i)| \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Therefore:

$$\prod_{v \in V(\mathcal{G})} |D(v)| = \prod_i |D(v_i)| \cdot \prod_{i,j} |D(u_i^j)| \geq \left(\prod_i |D(v_i)| \right)^{d+1} \geq (\frac{n}{2})^{\frac{(d+1)\cdot n}{2}}$$

If we take $\log_2$ of this number (to get the number of Boolean variables needed), we have $\frac{1}{2} \cdot (d+1) \cdot n \cdot (\log(n) - 1)$. In comparison, the following dynamic range is adequate for $\mathcal{G}$:

1. $v_i = \{1, \ldots, i\}$
2. $u_i^j = \{v_i\}$

Which is of size $n!$, i.e., $O(n \log(n))$ Boolean variables, which does not depend on d at all.

Note that the lower bound on the size of any static range is true for any E-graph containing this E-graph as a sub-graph.

10 Experimental Results

To test our dynamic range allocation procedure and compare our results to the static range allocation procedure of [PRSS98], we generated many random E-graphs. For each E-graph $\mathcal{G}$ we calculated the size of the resulting state space generated $|S|$, and then calculated $|S|^{\frac{1}{|V(G)|}}$ to give the average size of the range for each variable.

We believe dynamic range allocation performs especially well on E-graphs that can be divided into components with a small number of equality edges between them. We performed the following set of tests:

Set $V(\mathcal{G}) = \{1, \ldots, 100\}$. For each $p \in \{.2, .25, \ldots, 1\}$ and $q \in \{.01, .02, \ldots, .2\}$ we generate 10 random E-graphs, by letting each edge (i, j) be chosen with probability p if $i \ mod \ 4 = j \ mod \ 4$, and with probability q otherwise. This way we get 4 components with edge probability p for edges internal to these components, and probability q for edges between the components. We also ran a simpler set of tests, where for each $p \in \{.02, .04, \ldots, 1\}$ we generated 10 random E-graphs on 100 vertices, where each edge is taken with probability p.

In Figure 3 we see the summary of the results, where for each test we averaged the results ($|S|^{\frac{1}{100}}$) of dynamic and static ranges over the 10 graphs, and give

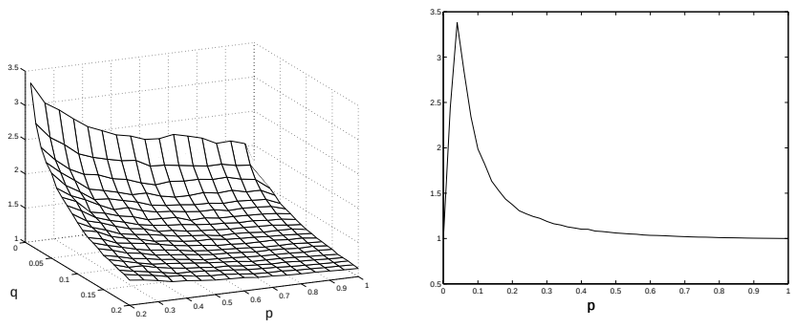


Fig. 3. The ratio between the average domain sizes (of each variable) allocated by the static and dynamic range allocation methods, as computed on graphs with 100 vertices and 4 components on the left graph, where p and q are the probabilities of adding edges in and between the components, respectively. The right graph summarizes the results for graphs with 100 vertices and edge probability p.

the ratio between them. We can see that for all cases the ratio is greater than 1, meaning the dynamic range allocation is at least as good (on the 10 graph average), and that on sparse graphs (either q is small or p is small) we get an improvement of approximately 2, which means a decrease of 2^{100} in the state space size. We have also implemented dynamic range allocation as a part of a procedure for checking uninterpreted functions [RS01], and achieved a factor 2 improvement in the range size, similar to the results on random graphs. However, we have not found a case where this improvement led to a significant change in running times. This is especially because our examples are of two types: the run either completes in less than 1 second, or it never completes.

References

BD94. J.R. Burch and D.L. Dill, "Automatic Verification of Microprocessor Control", In *Computer-Aided Verification CAV '94*.

BDL96. Clark W. Barrett, David L. Dill and Jeremy R. Levitt, "Validity Checking for Combinations of Theories with Equality", In *Formal Methods in Computer Aided Design FMCAD '96*.

BV00. R. E. Bryant and M. N. Velev, "Boolean satisfiability with transitivity constraints", In *Computer-Aided Verification CAV 2000*.

GSZAS98. A. Goel, K. Sajid, H. Zhou, A. Aziz and V. Singhal, "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions", In *Computer-Aided Verification CAV '98*.

PRSS98. A. Pnueli, Y. Rodeh, M. Seigel and O. Shtrichman, "Deciding Equality Formulas by Small Domain Instantiations", In *Computer-Aided Verification CAV '99*.

PSS98. A. Pnueli, M. Siegel and O. Shtrichman, "Translation Validation for Synchronous Languages", In *International Colloquium on Automata, Languages and Programming ICALP '98*.

RS01. Y. Rodeh and O. Shtrichman, "Finite Instantiations in Equivalence Logic with Uninterpreted Functions", In *Computer-Aided Verification CAV '01.*

A Proofs

A.1 Dynamic Range Allocation

Proof. (of Theorem 1): For a satisfiable $\mathcal{H} \leq \mathcal{G}$, we find $x \in D'$ such that $x \models \mathcal{H}$. We first notice that since $\mathcal{H}$ is satisfiable, $\mathcal{H}[v]$ is also satisfiable. Also, $\mathcal{H}[v] \leq \mathcal{G}[v]$. Therefore there is some $x \in D$ such that $x \models \mathcal{H}[v]$. We extend x to v, and therefore only need to show that x satisfies all edges of $\mathcal{H}$ involving v since all other edges are clearly satisfied by x.

1. If there is no $(u, v) \in EQ(\mathcal{H})$, then set $x(v) =$ **unique**. Since all edges involving v are disequality edges, we have that $x \models \mathcal{H}$.
2. If there is some $(u, v) \in EQ(\mathcal{H})$, then set $x(v) = u$:
   (a) For a vertex w such that $(v, w) \in EQ(\mathcal{H})$, if $w = u$ then clearly, $\overline{x}(w) = \overline{x}(v)$. Otherwise, there is an equality edges $(u, w) \in EQ(\mathcal{H}[v])$, and therefore $\overline{x}(w) = \overline{x}(u)$, and then $\overline{x}(w) = \overline{x}(v)$.
   (b) For a vertex w such that $(v, w) \in DQ(\mathcal{H})$, since $\mathcal{H}$ is satisfiable, $w \neq u$. Also, there is a disequality edge $(w, u) \in DQ(\mathcal{H}[v])$, and then $\overline{x}(v) = \overline{x}(u) \neq \overline{x}(w)$.

□

A.2 One-Orientable Assignment Sets

To prove proposition 1, we need the following definitions:

Definition 12. (forest): *A* forest *is an acyclic undirected graph.*

Definition 13. (spanning forest): *A* spanning forest *for a graph G, is a subgraph F of G, such that F is a forest, and the connected components of F and G are the same.*

Claim. Every graph has a spanning forest.

Definition 14. (forest partition set): *The* forest partition set *of the E-graph $\mathcal{G}$ is:*

$$F(\mathcal{G}) = \{\alpha_F \mid F \text{ is a forest and } F \leq \mathcal{G}_=\}$$

Proposition 4. *$F(\mathcal{G})$ is adequate for $\mathcal{G}$.*

Proof. Given a satisfiable $\mathcal{H} \leq \mathcal{G}$, take the forest F to be a spanning forest of the graph $\mathcal{H}_=$. Clearly, $E(F) \subseteq EQ(\mathcal{H}) \subseteq EQ(\mathcal{G})$. So, $\alpha_F \in R$.

We claim that $\alpha_F \models \mathcal{H}$:

1. If $(u, v) \in EQ(\mathcal{H})$, then u and v are in the same connected component of $\mathcal{H}_=$, and therefore of F. This means that $u \sim_{\alpha_F} v$.

2. If $(u, v) \in DQ(\mathcal{H})$ then u and v are not in the same connected component of $\mathcal{H}_=$ by Lemma 1. This means that they are in different components of F, and therefore $u \not\sim_{\alpha_F} v$.

□

Proof. (of Proposition 1): Since every forest is one-orientable (by rooting each of the trees in the forest, and directing all edges towards the root), we get that $F(\mathcal{G}) \subseteq One(\mathcal{G})$ and therefore $One(\mathcal{G})$ is adequate for $\mathcal{G}$. □

Proof. (of Proposition 2): We first show that the graphs represented by the variables $e_{(u,v)}$ are all the one-orientable sub-graphs of $\mathcal{G}_=$ possible.

Take some one-orientable $H \leq \mathcal{G}_=$. Denote by D the directed graph resulting from directing H's edges in such a way that every vertex has out-degree at most 1 in D. We define assignment a to the input variables of C. For each v:

- If there is exactly one u such that $(v, u) \in D$, set $a(I_v) = u$.
- Otherwise, there are no outgoing edges from v in D. Set $a(I_v) = \star$.

We get that $a(e_{(u,v)}) = 1$ iff $(u, v) \in H$. Therefore, $a(t_{(u,v)}) = 1$ iff u and v are connected in H. In other words $a(t_{(u,v)}) = 1$ iff $u \sim_{\alpha_H} v$.

So, for every $\alpha \in One(\mathcal{G})$, there is some assignment a to C, such that $a(C) = \varphi(\alpha)$, and since $One(\mathcal{G})$ is adequate for φ, we conclude. □

Rewrite Closure for Ground and Cancellative AC Theories*

Ashish Tiwari

SRI International,
333 Ravenswood Ave,
Menlo Park, CA, U.S.A
tiwari@csl.sri.com

Abstract. Given a binary relation $I\!E \cup I\!R$ on the set of ground terms over some signature, we define an abstract rewrite closure for $I\!E \cup I\!R$. An abstract rewrite closure can be interpreted as a specialized ground tree transducer (pair of bottom-up tree automata) and can be used to efficiently decide the reachability relation $\rightarrow^*_{I\!E \cup I\!E^- \cup I\!R}$. It is constructed using a completion like procedure. Correctness is established using proof ordering techniques. The procedure is extended, in a modular way, to deal with signatures containing cancellative associative commutative function symbols.

1 Introduction

Completion techniques for term rewriting systems, which are typically used for reasoning about congruence relations, have been extended in recent years to deal with non-symmetric relations. The general theory was outlined in [11] and sound and refutationally complete inference systems were obtained for dealing with partial congruence and partial equivalence relations [4]. Usually one obtains suitably restricted (via ordering restrictions) chaining calculi. The gain in efficiency with an ordered system over the unordered variants of chaining are comparable to the improvements achieved by superposition over unrestricted paramodulation.

This paper presents a completion based approach to *decide* the rewrite relation induced by a set of directed (i.e., non-symmetric) and undirected (i.e., symmetric) ground equations. The basic technique involves combining standard completion (for undirected equations) with non-symmetric completion (for directed equations). Standard completion is reflected in a *superposition* inference rule that deduces critical pairs between undirected equations. Non-symmetric completion yields a *chaining* inference rule to deduce critical pairs between directed equations. Finally, the interaction between the two kinds of equations is captured using a *paramodulation* inference rule. We first consider the problem of

* This research was supported in part by the National Science Foundation under grants CCR-9902031 and CCR-0082560, and NASA Langley Research Center under contract NAS1-00079.

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 334–346, 2001.
© Springer-Verlag Berlin Heidelberg 2001

constructing a "convergent system", called a *rewrite closure*, for a set of ground (directed and undirected) equations. Subsequently, we extend the method to allow for cancellative associative commutative function symbols in the signature. If all input equations are undirected, then the problem reduces to the construction of congruence closure and hence, an abstract rewrite closure is a generalization of an abstract congruence closure [6].

The reachability or rewrite relation induced by a ground term rewriting system was shown to be decidable in [9] and [13] using, respectively, tree automata techniques and explicit transitive closure computation. An abstract rewrite closure can be interpreted as a specialized "ground tree transducer" (GTT). In this paper, we give a set of abstract completion-like inference rules for construction of rewrite closures. These rules yield efficient algorithms under suitable strategies. Moreover, our method is extendible to richer signatures.

Correctness of the inference system is established using proof ordering techniques. Each proof is assigned a measure and all inference rules transform a proof with a larger measure into a proof with a smaller measure. The desired form of proof, for example a rewrite proof or a valley proof, is assigned a minimal measure. Correctness arguments based on proof orderings also show compatibility of the inference systems with certain kinds of simplifications.

Apart from our interest in extending rewriting techniques to non-symmetric relations, this work is also motivated by our interest in developing abstract transformation rules for constraint solving. Typical constraints consist of equational constraints, which are solved by a unification procedure, and ordering constraints, where the ordering is usually some kind of a path ordering. Almost all such orderings are rewrite relations that also satisfy certain additional properties, and hence an efficient procedure for deciding rewrite relations is a crucial first step [13]. Note that the cancellative axiom for AC symbols is satisfied by any AC compatible total simplification ordering.

Preliminaries

Let Σ be a set, called a *signature*, with an associated *arity function* $\alpha : \Sigma \to 2^{\mathbb{N}}$ and let $\mathcal{V}$ be a disjoint (denumerable) set. We define $\mathcal{T}(\Sigma, \mathcal{V})$ as the smallest set containing $\mathcal{V}$ and such that $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ whenever $f \in \Sigma, n \in \alpha(f)$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$. The elements of the sets Σ, $\mathcal{V}$ and $\mathcal{T}(\Sigma, \mathcal{V})$ are respectively called *function symbols*, *variables* and *terms* (over Σ and $\mathcal{V}$). Elements c in Σ for which $\alpha(c) = \{0\}$ are called *constants*. By $\mathcal{T}(\Sigma)$ we denote the set $\mathcal{T}(\Sigma, \emptyset)$ of all variable-free, or *ground* terms. The symbols $s, t, u, \ldots$ are used to denote terms; $f, g, \ldots$, function symbols; and $x, y, z, \ldots$, variables.

An *(undirected) equation* is an unordered pair of terms, written $s \approx t$. A *directed equation* or *rule* is an ordered pair of terms, written $s \to t$. If $\mathcal{E}$ is a set of rules, then we define $\mathcal{E}^- = \{s \to t : t \to s \in \mathcal{E}\}$ and $\mathcal{E}^{\pm} = \mathcal{E} \cup \mathcal{E}^-$. A set $\mathcal{E}$ of rules is called a *rewrite system* and the *rewrite relation* $\to_{\mathcal{E}}$ induced by $\mathcal{E}$ is defined by: $u \to_{\mathcal{E}} v$ if, and only if, $u = u[l\sigma]$, $v = u[r\sigma]$ is obtained by replacing $l\sigma$ by $r\sigma$ in u, $l \to r$ is in $\mathcal{E}$, and σ is some substitution. If $\to$ is a binary relation, then $\leftarrow$ denotes its inverse, $\leftrightarrow$ its symmetric closure, $\to^+$ its transitive closure

and $\rightarrow^*$ its reflexive-transitive closure. A set of rules $\mathcal{E}$ is *terminating* if there exists no infinite reduction sequence $s_0 \rightarrow_{\mathcal{E}} s_1 \rightarrow_{\mathcal{E}} s_2 \cdots$ of terms.

We will mostly be interested in ground rewrite systems, denoted by non-calligraphic symbols $I\!E, I\!R$. In Section 2, the arity $\alpha(f)$ of a symbol $f \in \Sigma$ is assumed to a singleton and we focus on (the transitive closure of) the rewrite relation $\rightarrow^*_{I\!E^{\pm} \cup I\!R}$ induced by the ground rewrite system $I\!E \cup I\!R$ over such a signature Σ. In Section 3, we shall assume that $\Sigma_{AC} \subset \Sigma$ is a set of AC symbols. Such symbols are varyadic, with arity $\alpha(f) = \{2, 3, 4, \ldots\}$ for $f \in \Sigma_{AC}$. If $f \in \Sigma_{AC}$, then the *extension* of a rule $f(s_1, s_2) \rightarrow t$, call it ρ, is defined as $f(f(s_1, s_2), x) \rightarrow f(t, x)$ and is denoted by ρ^e. Given a rewrite system $\mathcal{R}$, by $\mathcal{R}^e$ we denote the set $\mathcal{R}$ plus extensions of rules in $\mathcal{R}$. By $AC \backslash \mathcal{R}$ we denote the rewrite system consisting of all rules $u \rightarrow v$ such that $u \leftrightarrow^*_{AC} u'\sigma$ and $v = v'\sigma$, for some rule $u' \rightarrow v'$ in $\mathcal{R}$ and some substitution σ.

A *proof* of $s \rightarrow t$ (in $\mathcal{E}$) is a finite sequence $s = s_0 \rightarrow_{\mathcal{E}} s_1$, $s_1 \rightarrow_{\mathcal{E}} s_2$, $\cdots, s_{k-1} \rightarrow_{\mathcal{E}} s_k = t$ $(k \geq 0)$, which is usually written in abbreviated form as $s = s_0 \rightarrow_{\mathcal{E}} s_1 \rightarrow_{\mathcal{E}} \cdots \rightarrow_{\mathcal{E}} s_k = t$ $(k \geq 0)$.

2 Abstract Rewrite Closure

We closely follow the idea of an abstract congruence closure [6] in defining the notion of an abstract rewrite closure. More specifically, we *flatten* out terms via introduction of new constants and corresponding definitions.

Definition 1. *Let Σ be a signature and K be a set of constants disjoint from Σ. A* D-rule *(with respect to Σ and K) is a rewrite rule of the form $f(c_1, \ldots, c_k) \rightarrow c$ where $f \in \Sigma$ is a k-ary function symbol and $c_1, \ldots, c_k, c$ are constants in set K. A rewrite rule of the form $c \rightarrow f(c_1, \ldots, c_k)$ will be called a* reverse D-rule.

A C-rule *(with respect to K) is a rule $c \rightarrow d$, where c and d are constants in K.*

A set of D-rules and C-rules (with respect to Σ and K) is a specification of a bottom-up tree automaton transitions [8]. The set K represents "states" in the tree automaton. Thus, D-rules and C-rules represent regular and ϵ-transitions respectively. A set of ground equations and rules, say $I\!I_0 = I\!E_0 \cup I\!R_0$, where $I\!E_0 = \{f(g(a, b), g(a, b)) \approx a\}$ and $I\!R_0 = \{a \rightarrow b\}$, can be represented as $I\!I_1 = \{f(c_3, c_3) \approx c_1,\ c_1 \rightarrow c_2\}$ by introducing the set $E_1 = \{a \rightarrow c_1, b \rightarrow c_2, g(c_1, c_2) \rightarrow c_3\}$ of D-rules.

A constant c in K is said to *represent* a term t in $\mathcal{T}(\Sigma)$ via the rewrite system E if $t \leftrightarrow^*_E c$. For example, the constant c_3 represents the term $g(a, b)$ via E_1.

Definition 2 (Abstract rewrite closure). *Let Σ be a signature and K be a set of constants disjoint from Σ. A ground rewrite system $E \cup F \cup B$ is said to be an* (abstract) rewrite closure *(with respect to Σ and K) if*

(i) E and F are both sets of D-rules and C-rules, B is a set of reverse D-rules and C-rules such that each constant $c \in K$ represents some term $t \in \mathcal{T}(\Sigma)$ via E, and

(ii) the rewrite systems $E \cup F$ *and* $E \cup B^-$ *are terminating; and for all terms* $s, t \in \mathcal{T}(\Sigma)$*, if* $s \rightarrow^*_{E^{\pm} \cup F \cup B} t$ *then* $s \rightarrow^*_{E \cup F} \circ \leftarrow^*_{E \cup B^-} t$.
Moreover, if $\mathit{II} = \mathit{IE} \cup \mathit{IR}$ *is a set of ground equations and rules over* $\mathcal{T}(\Sigma)$ *such that*

(iii) for all terms s *and* t *in* $\mathcal{T}(\Sigma)$, $s \rightarrow^*_{\mathit{IE}^{\pm} \cup \mathit{IR}} t$ *if and only if* $s \rightarrow^*_{E^{\pm} \cup F \cup B} t$, *then* $E \cup F \cup B$ *will be called an (abstract) rewrite closure* for *(the rewrite relation induced by)* II.

From the set $E \cup F \cup B$, one can obtain a pair of (bottom-up) tree automata [8]: the set $E \cup F$ defines the transitions of the first automaton and the set $E \cup B^-$ defines the transitions of the second automaton (over the same set K of "states"). Such a pair defines a binary relation $\rightarrow^*_{E \cup F} \circ \leftarrow^*_{E \cup B^-}$ on the set of ground terms and is called a "ground tree transducer" in the tree automata literature [8].

Using a combination of standard completion and non-symmetric completion, which we present next, we can obtain a rewrite closure $E_2 \cup F_2 \cup B_2$ for the set $\mathit{II}_0 = \mathit{IE}_0 \cup \mathit{IR}_0$, where $E_2 = \{a \rightarrow c_1,\ b \rightarrow c_2,\ g(c_1, c_2) \rightarrow c_3,\ f(c_3, c_3) \rightarrow c_1\}$, $F_2 = \{c_1 \rightarrow c_2\}$, and $B_2 = \{c_3 \rightarrow g(c_2, c_2)\}$. A rewrite closure for $\mathit{IE} \cup \mathit{IR}$ gives a decision procedure for (deciding) the rewrite relation $\rightarrow^*_{\mathit{IE}^{\pm} \cup \mathit{IR}}$.

Construction of Rewrite Closure

We next present an inference system to construct a rewrite closure for a finite set II of ground equations and rules over the signature Σ. Our description is fairly abstract, in terms of transition rules that operate on tuples (II, E, R), where $\mathit{II} = \mathit{IE} \cup \mathit{IR}$ is a set of ground equations and rules (over Σ), and E and R[1] are sets of (reverse) D-rules and C-rules. Tuples represent possible *states* in the process of constructing a rewrite closure. The initial state is $(\mathit{II}_0, \emptyset, \emptyset)$, where II_0 is the input set of ground equations and rules (over $\mathcal{T}(\Sigma)$).

The transition rules can be derived from those for standard completion and non-symmetric completion as described in [3] and [11], with some differences so that a system is constructed over an *extended* signature. We assume that the new constants are chosen from an infinite set U disjoint from Σ, which is endowed with an ordering[2] $\succ_U$.

Equations and rules are flattened using extension and simplification.

Extension:
$$\frac{(\mathit{II}[s], E, R)}{(\mathit{II}[c], E \cup \{s \rightarrow c\}, R)}$$

if $s \rightarrow c$ is a D-rule and $c \in U$ is a new constant[3].

Simplification1:
$$\frac{(\mathit{II}[s], E \cup \{s \rightarrow c\}, R)}{(\mathit{II}[c], E \cup \{s \rightarrow c\}, R)}$$

[1] The set R will later be partitioned into the set F of forward rules and the set B of backward rules.

[2] By an *ordering* we mean any irreflexive and transitive relation on terms.

[3] The notation $\mathit{II}[s]$ denotes that s occurs as a subterm in some equation or rule in II and $\mathit{II}[c]$ denotes the new set obtained by replacing that occurrence of s in II by c.

Once an equation or rule in $\mathcal{I}$ is of the form of a D-rule, reverse D-rule, or a C-rule, it can be oriented.

$$\textbf{Ori}\text{entation:} \quad \frac{(\mathcal{I} \cup \{s \approx c\}, E, R)}{(\mathcal{I}, E \cup \{s \to c\}, R)} \qquad \frac{(\mathcal{I} \cup \{u \to v\}, E, R)}{(\mathcal{I}, E, R \cup \{u \to v\})}$$

if $s \to c$ is either a D-rule or a C-rule with $s \succ_U c$ and $u \to v$ is either a C-rule, D-rule, or a reverse D-rule.

Trivial equations and rules are deleted.

$$\textbf{Del}\text{etion:} \quad \frac{(\mathcal{I} \cup \{s \approx s\}, E, R)}{(\mathcal{I}, E, R)} \quad \frac{(\mathcal{I} \cup \{s \to s\}, E, R)}{(\mathcal{I}, E, R)} \quad \frac{(\mathcal{I}, E, R \cup \{s \to s\})}{(\mathcal{I}, E, R)}$$

Deduction in standard completion, as well as in non-symmetric completion, is based on computation of critical pairs. There are three kinds of critical pair computations—(i) between two rules in E, which are handled by superposition; (ii) between a rule in E and a rule in R, which are handled by paramodulation; and (iii) between two rules in R, which are handled by chaining.

$$\textbf{Sup}\text{erposition:} \quad \frac{(\mathcal{I}, E \cup \{t \to c, s[t] \to d\}, R)}{(\mathcal{I}, E \cup \{t \to c, s[c] \to d\}, R)} \quad \frac{(\mathcal{I}, E \cup \{t \to c, t \to d\}, R)}{(\mathcal{I}, E \cup \{t \to c, d \to c\}, R)}$$

if $s[t] \neq t$ in the first case and $d \succ_U c$ in the second case.

The set R can be partitioned into the set $F = \{s \to t \in R : s \to t$ is a D-rule or a C-rule with $s \succ_U t\}$ of *forward* rules and the set $B = \{s \to t \in R : t \to s$ is a D-rule or a C-rule with $t \succ_U s\}$ of *backward* rules.

Definition 3. *Let E and $R = F \cup B$ be sets of (reverse) D-rules and C-rules. The set $CP(E, R)$ of critical pairs between rules in E and R is defined as:*

$$\begin{aligned} CP(E,R) = \{&f(\ldots, d, \ldots) \to c : f(\ldots, d', \ldots) \to c \in E \text{ and } d \to d' \in B\} \\ \cup \{&c \to f(\ldots, d, \ldots) : f(\ldots, d', \ldots) \to c \in E \text{ and } d' \to d \in F\}. \end{aligned}$$

The set $CP(R)$ of critical pairs between rules in R is defined as:

$$\begin{aligned} CP(R) = \{&t[d] \to c : d \to s \in B \text{ and } t[s] \to c \in F\} \\ \cup \{&c \to t[d] : c \to t[s] \in B \text{ and } s \to d \in F\}. \end{aligned}$$

Note that if the sets E and R contain only D-rules, reverse D-rules, and C-rules, then so do the sets $CP(E, R)$ and $CP(R)$.

$$\textbf{Cha}\text{ining and } \textbf{Par}\text{amodulation:} \quad \frac{(\mathcal{I}, E, R)}{(\mathcal{I}, E, R \cup \{s \to t\})}$$

if $s \to t \in CP(R) \cup CP(E, R)$.

A crucial component of deductive inference systems is simplification. In the ground case, several deduction steps reduce to simplification. In particular, the rules in E can be used to simplify terms in R.

Simplification2:
$$\frac{(\mathit{I\!I}, E \cup \{s \to c\}, R[s])}{(\mathit{I\!I}, E \cup \{s \to c\}, R[c])}$$

Composition:
$$\frac{(\mathit{I\!I}, E \cup \{c \to d, s \to c\}, R)}{(\mathit{I\!I}, E \cup \{c \to d, s \to d\}, R)}$$

Example 1. Consider the set $\mathit{I\!I}_0 = \{f(g(a,b), g(a,b)) \approx a, a \to b\}$ of equations and rules. An abstract rewrite closure for $\mathit{I\!I}_0$ can be derived from $(\mathit{I\!I}_0, E_0, R_0) = (\mathit{I\!I}_0, \emptyset, \emptyset)$ as follows (assuming $U = \{c_0, c_1, c_2, \ldots\}$ with $c_i \succ_U c_j$ for $i < j$):

i	Input $\mathit{I\!I}_i$	Equations E_i	Rules R_i	Transition Rule
0	$\mathit{I\!I}_0$	$\emptyset$	$\emptyset$	
1	$\{fgabgab \approx a\}$	$\{a \to c_1, b \to c_2\}$	$\{c_1 \to c_2\}$	$\mathbf{Ext}^2 \circ \mathbf{Ori}$
2	$\{fc_3c_3 \approx a\}$	$E_1 \cup \{gc_1c_2 \to c_3\}$	R_1	$\mathbf{Sim}^4 \circ \mathbf{Ext} \circ \mathbf{Sim}$
3	$\emptyset$	$E_2 \cup \{fc_3c_3 \to c_1\}$	R_1	$\mathbf{Sim} \circ \mathbf{Ori}$
4	$\emptyset$	E_3	$R_1 \cup \{c_3 \to gc_2c_2\}$	$\mathbf{Par}$

Since no further rules are added, the rewrite system $E_4 \cup F_4 \cup B_4$, where $F_4 = \{c_1 \to c_2\}$ and $B_4 = \{c_3 \to gc_2c_2\}$, is an abstract rewrite closure for $\mathit{I\!I}_0$.

Correctness

We use the symbol $\vdash$ to denote the one-step transition relation on states induced by the above transition rules. A *derivation* is a sequence of states $(\mathit{I\!I}_0, E_0, R_0) \vdash (\mathit{I\!I}_1, E_1, R_1) \vdash \cdots$. A derivation is said to be *fair* if any transition rule which is continuously enabled is eventually applied. The set E_∞ of *persisting* rules is defined as $\cup_i \cap_{j>i} E_j$; and similarly, $R_\infty = \cup_i \cap_{j>i} R_j$.

We shall prove that any fair derivation will only generate finitely many persisting rewrite rules in the second and third components.

Theorem 1. *Let $\mathit{I\!I}_0$ be a finite set of ground equations and rules. The set $E_\infty \cup R_\infty$ of persisting rules in any fair derivation starting from the state $(\mathit{I\!I}_0, \emptyset, \emptyset)$ is finite.*

Proof. Each inference step either reduces, or leaves unchanged, the number of Σ-symbols in the $\mathit{I\!I}$-component. The inference rule which introduces new constants, extension, always reduces this number. Therefore, it follows that the number of new constants introduced in any derivation is finite. Let this number be n.

If the maximum arity of any function symbol in Σ is c, then the number of distinct D-rules is bounded by $|\Sigma|n^{c+1}$ and the number of distinct C-rules is n^2. Consequently, the sets E_∞ and R_∞ are finite. ■

Theorem 2 (Soundness). *If* $(\mathbb{E}_0 \cup \mathbb{R}_0, E_0, R_0) \vdash (\mathbb{E}_1 \cup \mathbb{R}_1, E_1, R_1)$*, then, the rewrite relation induced by* $\mathbb{E}_1^{\pm} \cup \mathbb{R}_1 \cup E_1^{\pm} \cup R_1$ *is identical to the rewrite relation induced by* $\mathbb{E}_0^{\pm} \cup \mathbb{R}_0 \cup E_0^{\pm} \cup R_0$ *over the set* $\mathcal{T}(\Sigma \cup K_0)$ *of terms, where* $K_0 \subset U$ *is the set of new constants introduced until state* $(\mathbb{I}_0, E_0, R_0)$*.*

Proof Ordering. The correctness of the procedure will be established using proof simplification techniques, as described by Bachmair [1] and Bachmair and Dershowitz [2], but specialized to our case of standard and non-symmetric ground completion. Let $\succ$ be any reduction ordering[4] which contains $\succ_U$ and also orients D-rules from left to right. For instance, a recursive path ordering with an appropriate precedence on function symbols is such an ordering.

Let $s = C[u] \to C[v] = t$ be a proof step using the equation or rule $u \approx v \in \mathbb{E}^{\pm} \cup \mathbb{R} \cup E^{\pm} \cup R$. The complexity of this proof step is defined by

$$\begin{array}{ll} (\{s,t\},\bot,\bot,\bot) \ \texttt{if} \ u \approx v \in \mathbb{E}^{\pm} & (\{s,t\},\bot,\bot,\bot) \ \texttt{if} \ u \to v \in \mathbb{R} \\ (\{s\},u,\bot,t) \ \texttt{if} \ u \to v \in E & (\{s\},u,\top,t) \ \texttt{if} \ u \to v \in R, u \succ v \\ (\{t\},v,\bot,s) \ \texttt{if} \ u \to v \in E^{-} & (\{t\},v,\top,s) \ \texttt{if} \ u \to v \in R, v \succ u \end{array}$$

where $\bot$ and $\top$ are new symbols assumed to be minimum and maximum respectively. Tuples are compared lexicographically using the multiset extension of the ordering $\succ$ on terms in the first component, and the ordering $\succ$ in the second and fourth component. The complexity of a proof is the multiset of complexities of its proof steps. The multiset extension of the ordering on tuples yields a proof ordering, denoted by $\succ_P$. The ordering $\succ_P$ is well-founded as it is a lexicographic combination of well-founded orderings.

Lemma 1. *Suppose* $(\mathbb{I}_0, E_0, R_0) \vdash (\mathbb{I}_1, E_1, R_1)$*. If* π *is a ground proof,* $s_0 \to s_1 \to \cdots \to s_k$*, in* $\mathbb{E}_0^{\pm} \cup \mathbb{R}_0 \cup E_0^{\pm} \cup R_0$*, then there is a proof* π'*,* $s_0 = s'_0 \to s'_1 \to \cdots \to s'_l = s_k$*, in* $\mathbb{E}_1^{\pm} \cup \mathbb{R}_1 \cup E_1^{\pm} \cup R_1$*, such that* $\pi \succeq_{\mathcal{P}} \pi'$*.*

Proof. We need to check that each equation or rule in $(\mathbb{E}_0 - \mathbb{E}_1)^{\pm} \cup (\mathbb{R}_0 - \mathbb{R}_1)^{\pm} \cup (E_0 - E_1) \cup (R_0 - R_1)$ has a simpler proof in $\mathbb{E}_1^{\pm} \cup \mathbb{R}_1 \cup E_1^{\pm} \cup R_1$ for each transition rule application. The details can be found in [16].

For instance, consider the case of simplification2 inference rule where $s[u] \to t \in R_0$ is simplified to $s[v] \to t \in R_1$ by the rule $u \to v \in E_0$. The old proof $s[u] \to_{R_0} t$ is replaced by the new proof $s[u] \to_{E_1} s[v] \to_{R_1} t$. If $s[u] \succ t$, then the new proof is smaller because the rewrite step $s[u] \to_{R_0} t$ is more complex than (a) the proof step $s[u] \to s[v]$ in either the second component, if $s[u] \neq u$, or the third component; and (b) the proof step $s[v] \to t$ in the first component as $s[u] \succ s[v]$ and $s[u] \succ t$. Next suppose that $t \succ s[u]$. In this case, the old rewrite step $s[u] \to_{R_0} t$ is more complex than (a) the proof step $s[u] \to s[v]$ in the first component as $t \succ s[u]$; and (b) the proof step $s[v] \to t$ in the fourth component as $s[u] \succ s[v]$. ■

Theorem 3 (Completeness). *Let* $\mathbb{I}_0$ *be a finite set of equations and rules. If* $(\mathbb{I}_\infty, E_\infty, F_\infty \cup B_\infty)$ *is the persisting state of a fair derivation starting from*

[4] A *reduction ordering* is an ordering that is well-founded and closed under contexts.

$(\mathbb{I}_0, \emptyset, \emptyset)$, then, the rewrite system $E_\infty \cup F_\infty \cup B_\infty$ is an abstract rewrite closure for $\mathbb{I}_0$.

Proof. (Sketch) Fairness implies that all superposition, paramodulation, and chaining inferences between rules in E_∞ and R_∞ are contained in the set $(\cup_i E_i) \cup (\cup_i R_i)$. Fairness also implies that $\mathbb{I}_\infty$ is empty. Since the proof ordering is well-founded, it follows from Lemma 1 that for every proof in $\mathbb{E}_i^{\pm} \cup \mathbb{R}_i \cup E_i^{\pm} \cup R_i$, there exists a minimal proof in $E_\infty^{\pm} \cup R_\infty$. We argue by contradiction that peaks, which are proof patterns of the form $s \to u \to t$ with $u \succ s$ and $u \succ t$, can not occur in the minimal proof. This implies that for all terms $s, t \in \mathcal{T}(\Sigma)$, if $s \to^*_{F_\infty \cup B_\infty \cup E_\infty^{\pm}} t$ then $s \to^*_{E_\infty \cup F_\infty} \circ \leftarrow^*_{E_\infty \cup B_\infty^-} t$. Moreover, the rewrite systems $E_\infty \cup F_\infty$ and $E_\infty \cup B_\infty^-$ are terminating as they are contained in $\succ$. Finally, property (i) and (ii) of Definition 2 follow from correctness of congruence closure [6] and Lemma 1. This establishes that $E_\infty \cup F_\infty \cup B_\infty$ is a rewrite closure *for* $\mathbb{E}_0 \cup \mathbb{R}_0$. ■

Related Work and Other Remarks

Note that the relation $\to^*_{E \cup F} \circ \leftarrow^*_{E \cup B^-}$ is decidable as the rewrite systems $E \cup F$ and $E \cup B^-$ are terminating [11]. Although the search for a proof of the above form involves guessing the correct rewrite rules to apply, we can still decide in polynomial time if $s \to^*_{E \cup F} \circ \leftarrow^*_{E \cup B^-} t$, as (i) the non-deterministic choices can be eliminated by maintaining subsets of K, that is, doing subset determinization along the computation, and (ii) the common context $C[_]$ of terms s and t such that $s \to^*_{E \cup F} C[c_1, \ldots, c_k] \leftarrow^*_{E \cup B^-} t$ can be determined by starting with the largest common context of s and t and moving (only polynomial number of times) to a smaller context if necessary. Furthermore, using the result that establishes a quadratic bound on the length of a derivation for construction of congruence closure [6], we can show that we can reach a state consisting of all persisting rules using derivations of length $O(n^2 + n^{c+1})$, where n is the size of the input and c is the maximum arity[5] of any symbol in Σ. Reachability for ground rewrite systems was shown to be decidable in polynomial time in [13].

The construction of abstract rewrite closure is similar to performing "iterative (or transitive) closure" on a ground tree transducer (representing the one-step rewriting relation). However, there are the following differences: (a) whereas a GTT is specified as a pair of bottom-up tree automata, an abstract rewrite closure has an additional component, E, which keeps track of the term representation[6] and the undirected equations, like $s \approx t$, in the input. Thus, the undirected equations are treated using congruence closure and not as two distinct rules, $s \to t$ and $t \to s$ (as would be done in the GTT approach); (b) our deduction rules are local and have ordering constraints. The computation of an iterative closure for GTT is done using exhaustive closure under the following rule (described in

[5] Without loss of generality, the maximum arity c can be treated as a constant.

[6] The D-rules in E (introduced by Extension) are interpreted as representing the term DAG [6].

our framework as): "deduce $c \to d$ if $f(c_1, \ldots, c_k) \to c \in E \cup B^-$, $f(c'_1, \ldots, c'_k) \to d \in E \cup F$, and for each i, c_i and c'_i represent some common term in $\mathcal{T}(\Sigma)$"[7]. In [13], *all* possible transitivity inferences are explicitly done; (c) our procedure is based on standard completion techniques and redundant inferences are avoided; (d) the correctness argument is in terms of proof orderings; and (e) our procedure can be extended to AC symbols, whereas tree automata techniques have not been extended to such richer signatures. We explain the last three points further below.

Correctness arguments based on proof orderings allow for clear identification of redundant inferences and compatible simplifications. To illustrate this point, consider an inference rule $(\mathit{I\!I}, E, R \cup \{s \to t, t \to s\}) \vdash (\mathit{I\!I}, E \cup \{s \to t\}, R)$, where $s \succ t$. This inference rule[8] is clearly sound. The completeness of the inference system that includes this rule easily follows by observing that the deleted rules, $s \to t$ and $t \to s$ in the R-component, have simpler proofs using the new rule in the E-component. The new proofs are simpler in the third component.

3 Ground Cancellative *AC* Theories

We next enrich the signature with additional AC symbols Σ_{AC}. Apart from the associative and commutative axioms, the symbols $f \in \Sigma_{AC}$ are assumed to satisfy the *cancellative* axioms (or inverse monotonicity axioms),

$$f(x_1, x_2, \ldots, x_m) \approx f(x_1, y_2, \ldots, y_m) \text{ iff } f(x_2, \ldots, x_m) \approx f(y_2, \ldots, y_m),$$
$$f(x_1, x_2, \ldots, x_m) \to f(x_1, y_2, \ldots, y_m) \text{ iff } f(x_2, \ldots, x_m) \to f(y_2, \ldots, y_m),$$

and the identity axiom $f(x, e_f) \approx x$, where e_f is the identity element for f.

In the presence of AC-symbols, apart from D-rules and C-rules, we additionally require A-rules of the form $f(c_1, c_2, \ldots, c_m) \to f(d_1, d_2, \ldots, d_k)$, where $m, k \in \alpha(f)$. Unlike D-rules and C-rules, A-rules do not correspond to any standard notion of a transition in bottom-up tree automata. The definition of a rewrite closure can be extended by allowing for A-rules and replacing standard rewriting by rewriting modulo AC [16].

We first consider the simple case of cancellative abelian monoid. Let signature $\Sigma = \Sigma_{AC} = \{\cdot\}$ and let $K = \{e, c_1, c_2, \cdots, c_m\}$ be a finite number of constants where e is an identity element for $\cdot$. We denote an application of $\cdot$ by juxtaposition and use exponentiation notation and write, for example, c_1^2 for the term $c_1 \cdot c_1$. Moreover, we denote by $[s, t]$ the term that is the greatest common divisor of s and t. Thus, $[c_1^2 c_2 c_3, c_1 c_2^2 c_4] = c_1 c_2$.

Let $R_0 = \{s_1 \to t_1, s_2 \to t_2, \ldots, s_n \to t_n\}$ be a set of directed rules over the signature $\Sigma \cup K$, where each rule $s_i \to t_i$ is (when fully flattened and reduced

[7] A stronger requirement (assuming each constant represents some term in $\mathcal{T}(\Sigma)$) is $c_i \leftrightarrow_C^* c'_i$, where C represents all the C-rules in $E \cup F \cup B$. This inference rule is similar in spirit to the inference rule used in Nelson-Oppen congruence closure algorithm [6].

[8] Having the rule $s \to t$ in E is advantageous as rules in E can be used for simplification (see the Simplification2 and Composition inference rules).

using the identity axiom) either a D-rule, a reverse D-rule, a C-rule, or an A-rule. We first show how to "complete" this set. We associate a measure with every rule. The measure will be a vector from the set N^n, where $N = \{0, 1, 2, \ldots\}$ is the set of natural numbers. For the initial set R_0 of rules, we assign measures as follows: the rule $s_i \to t_i$ is assigned the measure $\boldsymbol{\varepsilon_i} = \langle 0, \ldots, 0, 1, 0, \ldots, 0 \rangle$, where 1 is in exactly the i-th component.

We maintain the invariant that $[l_i, r_i] = e$ for all rules $l_i \to r_i \in R$ and hence, we assume that $[s_i, t_i] = e$ for every $i = 1, 2, \ldots, n$. Let $\succ$ be the lexicographic ordering, or the total degree lexicographic ordering [7].

ACC-Chaining:
$$\frac{(I\!I, E, R \cup \{s \to t, u \to v\})}{(I\!I, E, R \cup \{s \to t, u \to v, \frac{su}{[s,v][t,u]} \to \frac{tv}{[s,v][t,u]}\})}$$

if $[t, u] \neq e$ and either (a) $t \succ s$ and $u \succ v$; or (b) $s \succ t$, $u \succ v$, and $s \to t$ is not a C-rule; or (c) $t \succ s$, $v \succ u$, and $u \to v$ is not a C-rule. The new rule is assigned the measure $\boldsymbol{\alpha} + \boldsymbol{\beta}$, where $\boldsymbol{\alpha}$ is the measure associated with the rule $s \to t$ and $\boldsymbol{\beta}$ is the measure associated with the rule $u \to v$.

In order to ensure termination, we need to identify and delete redundant rules. The measure vector helps in doing this.

ACC-Collapse:
$$\frac{(I\!I, E, R \cup \{s \to t, u' \to v'\})}{(I\!I, E, R \cup \{s \to t, u \to v\})}$$

if $u' \leftrightarrow^*_{AC} su$, $v' \leftrightarrow^*_{AC} tv$, and $\boldsymbol{\alpha} < \boldsymbol{\beta}$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the measures associated with the rules $s \to t$ and $u' \to v'$ respectively. The rule $u \to v$ is assigned the measure $\boldsymbol{\beta} - \boldsymbol{\alpha}$.

Along with the Deletion3 rule, this forms a set of transformations that can be used to complete a given finite set of (reverse) D-, C-, and AC-rules over a signature $\Sigma = \Sigma_{AC}$ containing exactly one cancellative AC function symbol.

Example 2. Consider the set $R_0 = \{c_1^2 \to c_2, c_2^2 \to c_1\}$ of directed rules. We can complete this set as follows (we show only the third component of the state here as the other components remain unchanged):

i	Rules R_i	Meas	Inference
0	R_0		
1	$R_0 \cup \{c_1 c_2 \to e\}$	$[1, 1]$	$\mathbf{ACC - Ch}$
2	$R_1 \cup \{c_1^3 \to e\}$	$[2, 1]$	$\mathbf{ACC - Ch}$

i	Rules R_i	Meas	Inference
3	$R_2 \cup \{c_2^3 \to e\}$	$[1, 2]$	$\mathbf{ACC - Ch}$
4	$R_3 \cup \{c_1^2 c_2^2 \to e\}$	$[2, 2]$	$\mathbf{ACC - Ch}$
5	R_3		$\mathbf{ACC - Col}$

Any rule subsequently deduced by chaining can be simplified by collapse and no additional rules are added to the set R_3. Thus, the system R_3 is the desired completion.

We say $s \to^{[]}_R t$ if, and only if, there exists a term u such that $s \cdot u \to_R t \cdot u$. The reflexive-transitive closure of the relation $\to^{[]}_R$ and is denoted by $\to^{[*]}_R$.

Theorem 4 (Soundness). *Suppose $s \approx t \in R_i$, where $(\emptyset, \emptyset, R_i)$ is a state in any derivation starting from state $(\emptyset, \emptyset, R_0)$. Then, $s \to^{[*]}_{R_0 \cup AC} t$.*

Theorem 5 (Completeness). *Let R_0 be a finite set of (reverse) D-, C-, and AC-rules over $\Sigma \cup K$. The set R_∞ of persisting rules in any fair derivation starting from the state $(\emptyset, \emptyset, R_0)$ is finite. Furthermore, if $s \to^{[*]}_{R_0 \cup AC} t$, then there is a proof of the form $s \to^*_{AC \setminus F^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC \setminus B^{-e}} t$.*

We combine the inference rules for the individual cancellative AC symbols and the inference rules for uninterpreted ground terms to get a procedure for constructing a rewrite closure for a set of equations and rules over a signature containing cancellative AC function symbols [16]. There are a few technical difficulties here however. First, in the case of a monoid, the length of the measure vector assigned to a rule was determined by the number of rules in the initial R-component, R_0. In the general case, these rules are created by orientation and moved from the $\mathit{I\!I}$-component to the R-component. Secondly, in the case of a monoid, all the C-rules in the R-component had exactly one measure vector associated with them. In case of a signature with $|\Sigma_{AC}|$ AC symbols, each C-rule will have a measure vector associated with it for each $f \in \Sigma_{AC}$. Third, we need an AC-compatible ordering that orients the D-rules in the right way. For this purpose, we use the ordering $\succ$ defined in [15]. When comparing two terms from a monoid, it reduces to the total degree lexicographic ordering. Finally, we additionally need ACC-superposition and ACC-paramodulation rules, for details and correctness see [16].

Other Remarks. The equational theory induced by a set of ground equations over a signature containing (non-cancellative) AC-symbols can be conservatively represented by D-rules, C-rules, and A-rules [5]. But, if we are interested in the *rewrite* relation, then the problem becomes much harder, as classical petrinet reachability is equivalent to the decidability of the rewrite relation induced by a set of ground rules over an abelian semigroup. A derivation using the inference rules presented here does not converge in the case of abelian semigroups. For instance, consider the petrinet with two states c_1 and c_2 and two transitions $c_1^4 c_2 \to c_1 c_2^2$ and $c_1^4 c_2 \to c_1^3 c_2^2$. ACC-Chaining inferences (assuming a total degree lexicographic ordering with $c_1 \succ c_2$) yield infinitely many persisting rules $c_1^5 c_2 \to c_1 c_2^3$, $c_1^6 c_2 \to c_1 c_2^4, \ldots, c_1^n c_2 \to c_1 c_2^{n-2}$. The reachability problem for petri nets was shown to be decidable in [14, 12].

The problem of deciding reachability in the case of a cancellative monoid is related to solving a system of linear diophantine equations by "duality". Consider the system $\{4x_1 - x_2 - 2x_3 = 0, 3x_1 - 4x_2 + 5x_3 = 0\}$. This system can be transformed into the three rewrite rules $c_1^4 c_2^3 \to e$, $e \to c_1 c_2^4$, and $c_2^5 \to c_1^2$. The original system has a non-trivial solution if and only if $e \to^+ e$. The converse translation can be similarly done. This connection is not surprising since one motivation for considering the cancellative axiom for AC-symbols comes from AC-unification, where linear diophantine equations arise naturally.

4 Conclusion

We have presented a set of inference rules, derived from standard completion and non-symmetric completion, to construct a rewrite closure for a set of ground equations and rules over a signature that can possibly contain cancellative AC symbols. The procedure works over an extended signature, incorporates essential simplifications, and is terminating.

There are several directions in which we envisage future work. The inference rules can be extended by including rules for unification and for special kinds of rewrite relations, like the various path orderings. This would give abstract transformation rules for constraint solving. Another possible extension is to (obtain decision procedures for) ordered fields. In this context, the non-symmetric relation will be interpreted as the ordering relation $>$ on the field elements. This work can also be extended along the lines of tree automata techniques and could be used to obtain *efficient* decision procedures for several properties of ground rewrite systems, for example confluence.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments.

References

1. L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, Boston, 1991.
2. L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2 & 3):173–201, October 1989.
3. L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of the ACM*, 41:236–276, 1994.
4. L. Bachmair and H. Ganzinger. *Ordered chaining calculi for first-order theories of binary relations*. Technical Report MPI-I-95-2-009, 1995.
5. L. Bachmair, I.V.Ramakrishnan, A. Tiwari, and L. Vigneron. Congruence closure modulo associativity and commutativity. In *Frontiers of Combining Systems, 3rd Intl Workshop FroCoS 2000*, pages 245–259. Springer, 2000. LNAI 1794.
6. L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In D. McAllester, editor, *CADE*, pages 64–78. Springer, 2000. LNAI 1831. Full version to appear in J. of Automated Reasoning, `www.csl.sri.com/users/tiwari/`.
7. T. Becker and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, Berlin, 1993.
8. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at: `http://www.grappa.univ-lille3.fr/tata`, 1997.
9. M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of ground term rewriting systems. In *Proc IEEE Symposium on Logic in Computer Science, LICS*, pages 353–359. IEEE Computer Society Press, 1987.
10. C. Kirchner, editor. *Rewriting Techniques and Applications, RTA-93*, Montreal, Canada, 1993. Springer-Verlag. LNCS 690.

11. A. Levy and J. Agusti. Bi-rewriting, a term rewriting technique for monotone order relations. In Kirchner [10], pages 17–31. LNCS 690.
12. E. W. Mayr. Persistence of vector replacement systems is decidable. *Acta Informatica 15*, pages 309–318, 1981. NewsletterInfo: 8.
13. D. A. Plaisted. Polynomial time termination and constraint satisfaction tests. In Kirchner [10], pages 405–420. LNCS 690.
14. S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th annual ACM symposium on theory of computing*, pages 267–281, May 1982.
15. A. Rubio and R. Nieuwenhuis. A precedence-based total AC-compatible ordering. In Kirchner [10], pages 374–388. LNCS 690.
16. A. Tiwari. Rewrite closure for ground and cancellative AC theories. Available: `www.csl.sri.com/users/tiwari/fsttcs01.html`, 2001. Full version of this paper.

Author Index

Agrawal, Manindra 58, 70
Allender, Eric 1
Anderson, Richard 83
Arora, Sanjeev 16

Brim, Luboš 96

Calcagno, Cristiano 108
Černá, Ivana 96
Chen, Jianer 120
Chopra, Sumit 183

Dang, Zhe 132
Dold, Axel 144

Fisman, Dana 156
Forster, Jürgen 171
Friesen, Donald K. 120

Gopinath, K. 207
Gupta, Neelima 183
Guruswami, Venkatesan 195

Harel, David 18

Ibarra, Oscar H. 132

Jaggi, Neeraj 207
Jakoby, Andreas 219
Jia, Weijia 120

Kanj, Iyad A. 120
Kannan, Sampath 83
Karloff, Howard 83
Koren, Yehuda 18
Krause, Matthias 171
Krčál, Pavel 96
Krishnan, Radha 232

Ladner, Richard E. 83
Lokam, Satyanarayana V. 171
Lukovszki, Tamás 244

Madhusudan, P. 256
Maheshwari, Anil 244
Mastrolilli, Monaldo 268
Meenakshi, B. 256
Mubarakzjanov, Rustam 171

Nielsen, Mogens 280

O'Hearn, Peter W. 108

Pelánek, Radek 96
Peled, Doron 292
Plaku, Erion 305
Pnueli, Amir 156, 292, 317

Raghavachari, Balaji 232
Rodeh, Yoav 317

San Pietro, Pierluigi 132
Sassone, Vladimiro 280
Schindelhauer, Christian 219
Schmitt, Niels 171
Sen, Sandeep 183
Shparlinski, Igor E. 305
Shtrichman, Ofer 317
Simon, Hans Ulrich 171
Srba, Jiří 280
Stirling, Colin 42

Tiwari, Ashish 334

Vialard, Vincent 144

Yang, Hongseok 108

Zeh, Norbert 244
Zuck, Lenore 292
Zwick, Uri 57