

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Kees M. van Hee Rüdiger Valk (Eds.)

Applications and Theory of Petri Nets

29th International Conference, PETRI NETS 2008
Xi'an, China, June 23-27, 2008
Proceedings

Volume Editors

Kees M. van Hee
Technische Universiteit Eindhoven
Dept. Mathematics and Computer Science
HG 7.74, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
E-mail: k.m.v.hee@tue.nl

Rüdiger Valk
University Hamburg
Department of Computer Science
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
E-mail: valk@informatik.uni-hamburg.de

Library of Congress Control Number: 2008928015

CR Subject Classification (1998): F.1-3, C.1-2, G.2.2, D.2, D.4, J.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-68745-9 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-68745-0 Springer 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. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com

© Springer-Verlag Berlin Heidelberg 2008
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12277117 06/3180 5 4 3 2 1 0

Preface

This volume consists of the proceedings of the 29th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2008). The Petri Net conferences serve as annual meeting places to discuss the progress in the field of Petri nets and related models of concurrency. They provide a forum for researchers to present and discuss both applications and theoretical developments in this area. Novel tools and substantial enhancements to existing tools can also be presented. In addition, the conferences always welcome a range of invited talks that survey related domains, as well as satellite events such as tutorials and workshops. The 2008 conference had six invited speakers, two advanced tutorials, and four workshops. Detailed information about PETRI NETS 2008 and the related events can be found at <http://ictt.xidian.edu.cn/atpn-acsd2008>.

The PETRI NETS 2008 conference was organized by the Institute of Computing Theory and Technology at Xidian University, Xi'an, China, where it took place during June 23-27, 2008. We would like to express our deep thanks to the Organizing Committee, chaired by Zhenhua Duan, for the time and effort invested in the conference and for all the help with local organization. We are also grateful for the financial support of the National Natural Science Foundation of China (NSFC) (Grant No. 60433010), Xidian University, and the Institute of Computing Theory and Technology at Xidian University.

This year we received 75 submissions by authors from 21 different countries. We thank all the authors who submitted papers. Each paper was reviewed by at least four referees. The Program Committee meeting took place in Eindhoven, The Netherlands, and was attended by 20 Program Committee members. At the meeting, 23 papers were selected, classified as: theory papers (14 accepted), application papers (5 accepted), and tool papers (4 accepted). We wish to thank the Program Committee members and other reviewers for their careful and timely evaluation of the submissions before the meeting. Special thanks are due to Martin Karusseit, University of Dortmund, for his friendly attitude and technical support with the Online Conference Service. Finally, we wish to express our gratitude to the six invited speakers, Gustavo Alonso, Jifeng He, Mike Kishinevsky, Huimin Lin, Carl Adam Petri (presented by Rüdiger Valk), and Grzegorz Rozenberg for their contribution. As usual, the Springer LNCS team provided high-quality support in the preparation of this volume.

April 2008

Kees van Hee
Rüdiger Valk

Organization

Steering Committee

Will van der Aalst, The Netherlands
Jonathan Billington, Australia
Gianfranco Ciardo, USA
Jörg Desel, Germany
Susanna Donatelli, Italy
Serge Haddad, France
Kurt Jensen, Denmark (Chair)
Jetty Kleijn, The Netherlands

Maciej Koutny, UK
Sadatoshi Kumagai, Japan
Carl Adam Petri (Honorary Member)
Lucia Pomello, Italy
Wolfgang Reisig, Germany
Grzegorz Rozenberg, The Netherlands
Manuel Silva, Spain
Alex Yakovlev, UK

Organizing Committee

Zhenhua Duan (Chair)
Xinbo Gao
Cong Tian

Haibin Zhang
Chenting Zhao
Yuanyuan Zuo

Tool Demonstration

Xiaobing Wang (Chair)

Program Committee

Kamel Barkaoui, France
Simona Bernardi, Italy
Luca Bernardinello, Italy
Eike Best, Germany
Didier Buchs, Switzerland
José Manuel Colom, Spain
Raymond Devillers, Belgium
Zhenhua Duan, China
Jorge de Figueiredo, Brazil
Giuliana Franceschinis, Italy
Luis Gomes, Portugal
Boudewijn Haverkort,
The Netherlands
Kees van Hee,
The Netherlands (Co-chair)
Claude Jard, France

ChangJun Jiang, China
Gabriel Juhas, Slovak Republic
Peter Kemper, USA
Ekkart Kindler, Denmark
Victor Khomenko, UK
Jetty Kleijn, The Netherlands
Fabrice Kordon, France
Lars Michael Kristensen, Denmark
Charles Lakos, Australia
ZhiWu Li, China
Johan Lilius, Finland
Chuang Lin, China
Robert Lorenz, Germany
Junzhou Luo, China
Toshiyuki Miyamoto, Japan
Patrice Moreaux, France

Madhavan Mukund, India
 Wojciech Penczek, Poland
 Michele Pinna, Italy
 Zhiguang Shan, China
 Kohkichi Tsuji, Japan

Rüdiger Valk, Germany (Co-chair)
 Hagen Völzer, Switzerland
 Karsten Wolf, Germany
 MengChu Zhou, USA

Referees

Ilham Alloui
 Soheib Baarir
 Gianfranco Balbo
 Paolo Baldan
 Paolo Ballarini
 João Paulo Barros
 Marco Beccuti
 Matthias Becker
 Marek Bednarczyk
 Maurice ter Beek
 Mehdi Ben Hmida
 Béatrice Bérard
 Robin Bergenthum
 Filippo Bonchi
 Marcello Bonsangue
 Lawrence Cabac
 Josep Carmona
 Davide Cerotti
 Prakash Chandrasekaran
 Thomas Chatain
 Ang Chen
 Lucia Cloth
 Flavio De Paoli
 Jörg Desel
 Boudewijn van Dongen
 Mariagrazia Dotoli
 Yuyue Du
 Claude Dutheillet
 Emmanuelle Encrenaz
 Johan Ersfolk
 Sami Evangelista
 Eric Fabre
 Dirk Fahland
 Carlo Ferigato
 João M. Fernandes
 Hans Fleischhack
 Jörn Freiheit

Fabio Gadducci
 Michele Garetto
 Thomas Gazagnaire
 Qi-Wei Ge
 Roberto Gorrieri
 Luuk Groenewegen
 Susumu Hashizume
 Reiko Heckel
 Monika Heiner
 Kai-Steffen Hielscher
 Lom Hillah
 Kunihiro Hiraishi
 Hendrik Jan Hoozeboom
 Andras Horvath
 Steve Hostettler
 Matthias Jantzen
 Alain Jean-Marie
 YoungWoo Kim
 Michael
 Köhler-Bußmeier
 Christian Kölbl
 Matthias Kuntz
 Mirosław Kurkowski
 Kristian Bisgaard Lassen
 Fedor Lehocki
 Chen Li
 Jianqiang Li
 Didier Lime
 Ding Liu
 Kamal Lodaya
 Niels Lohmann
 Ricardo Machado
 Agnes Madalinski
 Thomas Mailund
 Mourad Maouche
 Marco Mascheroni
 Peter Massuthe

Ilaria Matteucci
 Sebastian Mauser
 Antoni Mazurkiewicz
 Agathe Merceron
 Roland Meyer
 Andrew S. Miner
 Daniel Moldt
 Arjan Mooij
 Rémi Morin
 Mourad Mourad
 Wojciech Nabialek
 Morikazu Nakamura
 K. Narayan Kumar
 Artur Niewiadomski
 Olivia Oanea
 Atsushi Ohta
 Emmanuel Paviot-Adet
 Laure Petrucci
 Axel Poigné
 Denis Poitrenaud
 Agata Pórola
 Lucia Pomello
 Ivan Porres
 Jean-François
 Pradat-Peyre
 Astrid Rakow
 Anne Remke
 Pierre-Alain Reynier
 Andrea Saba
 Ramin Sadre
 Nabila Salmi
 Carla Seatzu
 Matteo Sereno
 Zuzana Sevcikova
 Jeremy Sproston
 Marielle Stoelinga
 Ping Sun

S.P. Suresh	Laurent Van Begin	Oskar Wibling
Koji Takahashi	Eric Verbeek	Elke Wilkeit
Shigemasa Takai	Enrico Vicario	Harro Wimmel
Wei Tan	Valeria Vittorini	Józef Winkowski
Xianfei Tang	Marc Voorhoeve	Keyi Xing
P. S. Thiagarajan	Jean-Baptiste Voron	Shingo Yamaguchi
Ferucio Laurentiu Tiplea	Jiacun Wang	Satoshi Yamane
Simon Tjell	Jan Martijn van der Werf	Tatsushi Yamasaki
Toshimitsu Ushio	Michael Westergaard	Tadanao Zanma

Table of Contents

Invited Papers

Challenges and Opportunities for Formal Specifications in Service Oriented Architectures	1
<i>Gustavo Alonso</i>	
Modeling Interactions between Biochemical Reactions (Abstract)	7
<i>A. Ehrenfeucht and G. Rozenberg</i>	
Transaction Calculus (Abstract)	8
<i>He Jifeng</i>	
Stratifying Winning Positions in Parity Games	9
<i>Huimin Lin</i>	
On the Physical Basics of Information Flow: Results Obtained in Cooperation with Konrad Zuse	12
<i>Carl Adam Petri</i>	

Regular Papers

Faster Unfolding of General Petri Nets Based on Token Flows	13
<i>Robin Bergenthum, Robert Lorenz, and Sebastian Mauser</i>	
Decomposition Theorems for Bounded Persistent Petri Nets	33
<i>Eike Best and Philippe Darondeau</i>	
Compositional Specification of Web Services Via Behavioural Equivalence of Nets: A Case Study	52
<i>Filippo Bonchi, Antonio Brogi, Sara Corfini, and Fabio Gadducci</i>	
Modeling and Analysis of Security Protocols Using Role Based Specifications and Petri Nets	72
<i>Roland Bouroulet, Raymond Devillers, Hanna Klaudel, Elisabeth Pelz, and Franck Pommereau</i>	
A Symbolic Algorithm for the Synthesis of Bounded Petri Nets	92
<i>J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev</i>	
Synthesis of Nets with Step Firing Policies	112
<i>Philippe Darondeau, Maciej Koutny, Marta Pietkiewicz-Koutny, and Alex Yakovlev</i>	

Modelling and Analysis of the INVITE Transaction of the Session Initiation Protocol Using Coloured Petri Nets	132
<i>Lay G. Ding and Lin Liu</i>	
Modelling and Initial Validation of the DYMO Routing Protocol for Mobile Ad-Hoc Networks	152
<i>Kristian L. Espensen, Mads K. Kjeldsen, and Lars M. Kristensen</i>	
Formal Specification and Validation of Secure Connection Establishment in a Generic Access Network Scenario	171
<i>Paul Fleischer and Lars M. Kristensen</i>	
Parametric Language Analysis of the Class of Stop-and-Wait Protocols	191
<i>Guy Edward Gallasch and Jonathan Billington</i>	
Hierarchical Set Decision Diagrams and Automatic Saturation	211
<i>Alexandre Hamez, Yann Thierry-Mieg, and Fabrice Kordon</i>	
Performance Evaluation of Workflows Using Continuous Petri Nets with Interval Firing Speeds	231
<i>Kunihiko Hiraishi</i>	
Modelling Concurrency with Quotient Monoids	251
<i>Ryszard Janicki and Dai Tri Man Lê</i>	
Labeled Step Sequences in Petri Nets	270
<i>Matthias Jantzen and Georg Zetsche</i>	
MC-SOG: An LTL Model Checker Based on Symbolic Observation Graphs	288
<i>Kais Klai and Denis Poitrenaud</i>	
Symbolic State Space of Stopwatch Petri Nets with Discrete-Time Semantics	307
<i>Morgan Magnin, Didier Lime, and Olivier (H.) Roux</i>	
A Practical Approach to Verification of Mobile Systems Using Net Unfoldings	327
<i>Roland Meyer, Victor Khomenko, and Tim Strazny</i>	
Cooperative Arrival Management in Air Traffic Control - A Coloured Petri Net Model of Sequence Planning	348
<i>Hendrik Oberheid and Dirk Söffker</i>	
Process Discovery Using Integer Linear Programming	368
<i>J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik</i>	

Tool Papers

Synthesis of Petri Nets from Scenarios with VipTool	388
<i>Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser</i>	
A Monitoring Toolset for PAOSE	399
<i>Lawrence Cabac, Till Dörge, and Heiko Rölke</i>	
Animated Graphical User Interface Generator Framework for Input-Output Place-Transition Petri Net Models	409
<i>João Lourenco and Luís Gomes</i>	
HYPENS: A Matlab Tool for Timed Discrete, Continuous and Hybrid Petri Nets	419
<i>Fausto Sessego, Alessandro Giua, and Carla Seatzu</i>	
Author Index	429

Challenges and Opportunities for Formal Specifications in Service Oriented Architectures

Gustavo Alonso

Systems Group
Department of Computer Science
ETH Zurich, Switzerland
{alonso}@inf.ethz.ch
<http://www.inf.ethz.ch/~alonso>

Abstract. Service Oriented Architectures (SOA) are currently attracting a lot of attention in industry as the latest conceptual tool for managing large enterprise computing infrastructures. SOA is interesting from a research perspective for a variety of reasons. From the software engineering side, because it shifts the focus away from conventional programming to application integration, thereby challenging many of the premises around improving development by improving programming languages. From the middleware point of view, SOA emphasizes asynchronous interaction, away from the RPC/RMI model, and thus brings to the fore many of the inadequacies of existing software and hardware platforms. From the formal specification perspective, however, SOA offers many opportunities as one of the key ideas behind SOA is the notion of capturing the interactions at a high level and letting the underlying infrastructure take care of the implementation details. For instance, the emphasis in SOA is the main reason why workflow and business process technologies are experiencing a new revival, as they are seen as a way to formally specify complex interaction patterns. This presentation covers the main ideas behind SOA and why they are an excellent basis to provide a more formal basis for the development and evolution of complex systems.

Keywords: Multi-tier architectures, Service Oriented Architectures, Web Services, Workflow, Business Processes, Declarative Languages.

1 Introduction

Service Oriented Architectures (SOA) are the latest approach to deliver a better understanding and improved techniques to master the complexities of the modern enterprise architectures. SOA differs from past attempts in several fundamental ways. First, it is language independent and makes no assumption about the underlying programming model. Second, communication is no longer based almost exclusively on request-response patterns (RPC/RMI) but the emphasis is on asynchronous events and messages. Third, SOA sees the development of new applications and services mainly as the integration and composition of large

scale services and applications rather than as a smaller scale programming problem. These differences arise from the lessons learned in the last two decades and represent a significant step forward. These differences also create very interesting opportunities and pose a wide range of challenges that will be discussed in this presentation.

2 Background

2.1 Multi-tier Architectures

There is a general classification commonly used for multi-tier systems that is useful for discussing enterprise architectures [ACKM03]. It is based on a number of layers, each one build on top of the lower one and providing distinct functionality. The layers are logical entities and can be implemented in a variety of ways, not necessarily always as separate entities. The lowest layer I the classification is the Resource Manager layer, which provides all the data services necessary to implement the upper layers. The resource manager is often implemented as a database but it can also be simpler (e.g., a file system) or much more complex (e.g., an entire sub-systems also implemented as a combination of layers). The Application Logic is built directly on top and it contains the code that implements the functionality that defines the application. The application logic layer can be as simple as a monolithic program or as complex as a fully distributed application. The final layer is the Presentation Layer, or the part of the system that deal with external interactions, regardless of whether they are with users or with other applications. The presentation layer should not be confused with a client, which is a particular form of distributing the presentation layer. The presentation layer is not the final interface (e.g., a web browser) but the functionality that prepares the information to be sent outside the application (e.g., the web server). The properties of an enterprise architecture can be studied based on the integration patterns that arise as the different layers are combined and distributed in different forms. There are four basic patterns: one tier, two-tier, three tier, and multi-tier architectures.

One tier architectures arose historically from mainframes, where the applications where implemented in a monolithic manner without any true distinction between the layers. Such design can be highly optimized to suit the underlying hardware; the lack of context switches and network communication also removes a great deal of overhead compared with other architectures. An important characteristic of one tier architectures is that the presentation layer is part of the architecture and the clients are passive in the sense that they only display the information prepared before hand in the one tier architecture. Decades ago this was the way dumb-terminals operated. Today, this is the way browser based applications based on HTML work and the basis for the so called Software as a Service (SaaS) approaches, where the client does not run any significant part of the application. Although SaaS is not implemented as a one tier architecture on

the server side, the properties it exhibits from the outside are similar to those of a one tier architecture.

Two tier architectures take advantage of distribution by moving the presentation layer to the so called client while leaving the application logic and the resource manager in the so called server. The result is a system with two tier that are generally tightly coupled as the client is developed for a particular server, since the client is actually the presentation layer of the global application. Two tier architectures represented historically a very important departure in terms of technology as they appeared at the same time as RPC and led to the definition of many different standard interfaces. Yet, they face important practical limitations: if a client needs to be connected to different servers, the client becomes the point of integration. From an architectural point of view, this is not scalable beyond a few servers and creates substantial problems maintaining the software. On the server side, coping with an increasing number of clients is complicated if the connections are stateful and the server monolithic as it then becomes bottleneck.

In part to accommodate the drawbacks of client server architectures, and in part to provide the necessary flexibility to adapt to the Internet, two tier architectures eventually evolved into three tier architectures. The notion of middleware, as the part of the infrastructure that resides between the client and the server, characterizes three tier architectures quite well. The middleware provides a single interface to all the clients and to all the servers; it serves as the intermediate representation that removes the need for application specific representations; and becomes as supporting infrastructure to implement useful functionality that simplifies the development of the application logic. In its full generality, the middleware helps to connect a fully distributed infrastructure where not only every tier is distributed. Each tier can in turn be distributed itself as it happens today in large scale web server systems.

From three tier architectures, a variety of integration patterns have evolved in the last two decades that range from RPC based infrastructures (e.g., CORBA) to message oriented middleware (MOM) where the interactions are asynchronous and implemented through message exchanges rather than request/response operations. MOM was for a while a useful extension to conventional (RPC based) middleware. Slowly, however, developers and architects started to realize the advantages of message based interaction over the tight coupling induced by RPC systems. Today asynchronous interactions are a very important part of the overall architecture and, to a very large extent, have become a whole integration layer by themselves.

From an architectural point of view, few real systems match one of those architectural patterns exactly. In particular, complex systems are often referred to a N-tier architectures to reflect the fact that they are a complex combination of layered systems, distributed infrastructures, and legacy systems tied together often with several layers of integration infrastructure. These complex architectures are the target of this presentation as they form the basis of what is called an enterprise architecture.

2.2 Web Services and Service Oriented Architecture

Multi-tier architectures integrate a multitude of applications and systems into a more or less seamless IT platform. The best way to construct such systems is still more an art than a science although in the last years some structure has started to appear in the discussion. During the 90's middleware platforms proliferated, with each one of them implementing a given functionality (transactions, messaging, web interactions, persistence, etc.) [A04]. Most of these platforms had a kernel that was functionally identical to all of them, plus extensions that gave the system its individual characteristics. In many cases, these systems were used because of the functionality to all of them (e.g., TP-monitors that were used because they were some of the earliest and better multi-threading platforms). The extensions were also the source of incompatibilities, accentuated by a lack of standardization of the common aspects [S02].

The emphasis today has changed towards standardization and a far more formal treatment of the problems of multi-tier architectures. The notion of "enterprise architecture" is a direct result of these more formal approaches that start distinguishing between aspects such as governance, integration, maintenance, evolution, deployment, development, performance engineering, etc. Two of the concepts that dominate the discourse these days are web services and service oriented architectures. Although in principle they are independent of each other and they are not entirely uncontroversial, they are very useful stepping stones to understand the current state of the art.

Web services appeared for a wide variety of reasons, the most important ones being the need to interact through the Internet and the failure of CORBA to build the so called universal software bus [A04]. Once the hype and the exaggerated critique around web services are removed, there remain a few very useful concepts that are critical to enterprise architectures. The first of them is SOAP, a protocol designed to encapsulate information within an "envelope" so that applications can interact regardless of the programming language they use, the operating system they run on, or their location. SOAP is interesting in that it avoids the pitfalls of previous approaches (e.g., RPC, CORBA, RMI) by not making any assumptions about the sender and the receiver. It only assumes that they can understand one of several intermediate representations (bindings). The second important concept behind web services is WSDL, or the definition of services as the interfaces to complex applications. WSDL describes not only the abstract interface but also the concrete implementations that can be used to contact that service. By doing so it introduces the possibility of having a service that can be used over several protocols and through different representations, the basis for adaptive integration and an autonomic integration infrastructure. Web Services do not need to involve any web based communication. In fact, they are extremely useful in multi-tier architectures as the interfaces and basic communication mechanism across systems and tiers. Dynamic binding (such as that implemented in the Web Services Invocation Framework of Apache) allows to give a system a web service interface and lets the infrastructure choose the best possible binding depending on the nature of the caller. A local piece of code

will use a native protocol, a remote caller using the same language will use a language specific binding, a remote a heterogeneous caller will use an XML/SOAP binding. It is also possible to link request/response interfaces to message based interfaces and vice-versa, a clear indication of the potential of the ideas.

The notion of service used in Web Services [ACKM03] has proven to be very useful, more useful than the notions of components or objects used in the past. Services are large, heterogeneous, and autonomous systems offering a well defined interface for interacting with them. They might or might not be object oriented, and it is irrelevant how they are implemented. The key to access them is to support one of the bindings offered by the service. From this idea, it is only a short step to the notion of Service Oriented Architectures, i.e., enterprise architectures where the different elements are connected through service interfaces and where the interactions happen through the interfaces but preferably using asynchronous exchanges like those used in MOM systems. Thus, SOA is a new attempt at realizing the vision of the universal software bus that CORBA started. This time, however, with a much wider understanding of the problems and forcing the programming language concerns out of the picture. SOA is not about programming, like CORBA was (or is). SOA is about integration and about how to best implement architectures that are based on integrating software systems.

Web Services and SOA have been around for just a few years but they have brought with themselves enough interesting concepts that their ideas will become part of the solution, regardless of what form this solution eventually takes. In many ways, they are the first step towards eliminating "spaghetti integration" and moving towards more structured and modular approaches to application integration. In the same way that it was a long way from the initial structured programming languages to today's modern object oriented languages, SOA and Web Services are the first step towards better tools and a more sophisticated understanding of application integration.

SOA has allowed system architects to formulate many problems in a clearer manner than it was possible before. From the problem of defining services, to the types of interactions between these services, including the organization of services as well as the associated problems of ensuring quality of service from design/development time, SOA has helped developers to define a proper framework to tackle such problems in an orderly manner.

3 High Level Programming in SOA

This presentation argues that SOA provides an ideal starting point for a more formal approach to the design, development, deployment, and maintenance of complex, distributed, enterprise architectures. As mentioned in the abstract, SOA is the reason why workflow management systems and business processes are again in fashion. SOA proposes to work at the level of services: large scale applications hidden behind a service interface that can support asynchronous interactions. SOA is about how to build and connect those services into larger

entities. This is very similar to what business processes aim to do: combine a variety of applications into a coherent process that matches a given set of business requirements. It is not surprising then that SOA and business processes have by now become terms that are often mentioned together.

The presentation will provide a historical overview of how multi-tier architectures have evolved and how their evolution led to SOA and Web services. The discussion will include what are the significant differences between SOA and approaches like CORBA or general middleware. Based on this background (briefly explained as well in this extended abstract), the presentation will then focus on what are the advantages of SOA from a technical point of view as well as on the challenges and opportunities for using formal specifications in the context of SOA.

The opportunities SOA offers are cleaner interfaces, a wide range of technical solutions to solve the integration problem, and a cleaner architecture where integration is the primary goal. The challenges to solve before we can take advantage of these opportunities are the embedding of these ideas into conventional programming languages (or within more modern, more suitable languages), the complexity of asynchronous systems, and the need to set reasonable goals for the formal specifications.

References

- [ACKM03] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer, Heidelberg (2003)
- [A04] Alonso, G.: Myths around Web Services. In: Bulletin of the Technical Committee on Data Engineering, December 2002, vol. 25(4) (2002)
- [S02] Stonebraker, M.: Too much Middleware. Sigmod Record (March 2002)

Modeling Interactions between Biochemical Reactions

A. Ehrenfeucht¹ and G. Rozenberg^{1,2}

¹ Department of Computer Science, University of Colorado at Boulder,
Boulder, CO 80309, USA

² Leiden Institute of Advanced Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Abstract. From the information processing point of view a living cell consists of a (huge) number of biochemical reactions that interact with each other. Two main mechanisms through which biochemical reactions influence each other are facilitation and inhibition. Therefore it is reasonable to base a formal model of interactions between biochemical reactions on a suitable formalization of these two mechanisms. Recently introduction reaction systems is a formal model following this way of reasoning.

We have made a number of assumptions that hold for a great number of biochemical reactions and therefore underlie the model of reaction systems. The assumptions are:

- Reactions are primary, while structures are secondary.
- There is a “threshold supply” of elements: either an element is present and then there is “enough” of it, or an element is not present. Thus there is no counting in the basic model.
- There is no “permanency” of elements: if “nothing” happens to an element, then it ceases to exist. Sustaining an element requires an effort (“life/existence must be sustained”).

We will argue that assumptions underlying the functioning of biochemical reactions are very different to the underlying axioms of standard models in theoretical computer science (including the model of Petri nets).

Transaction Calculus

He Jifeng^{*}

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, China

Abstract. Transaction-based services are increasingly being applied in solving many universal interoperability problems. Exception and failure are the typical phenomena of the execution of long-running transactions. To accommodate these new program features, we extend the Guarded Command Language by addition of compensation and coordination combinators, and enrich the standard design model with new healthiness conditions. The paper proposes a mathematical framework for transactions where a transaction is treated as a mapping from its environment to compensable programs. We provide a transaction refinement calculus, and show that every transaction can be converted to primitive one which simply consists of a forward activity and a compensation module.

^{*} This work was supported by the National Basic Research Program of China (Grant No. 2005CB321904) and Shanghai Leading Academic Discipline Project B412.

Stratifying Winning Positions in Parity Games^{*}

Huimin Lin

Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
lhm@ios.ac.cn

Parity games have attracted considerable attentions for at least two reasons. First, the problem of deciding winning positions in parity games is equivalent to the problem of μ -calculus model checking. Second, parity games are the simplest in a chain of two-player games which are known in $\text{NP} \cap \text{co-NP}$ but so far no efficient algorithms have been found.

A parity game is played on a (finite) graph, referred to as *game graph*, by two players, Ms. Even and Mr. Odd. Each node n in the graph is assigned a *weight*, or *depth*, $D(n)$ which is a natural number, and is labelled by either $\vee$ or $\wedge$. The $\vee$ -nodes are Mr. Odd's positions, while the $\wedge$ -nodes are Ms. Even's. Without loss of generality we may assume a $\vee$ -node has either one or two out-going edges, and a $\wedge$ -node has exactly two out-going edges.

A play can be viewed as moving a token along the edges of the game graph. A token can be moved from a node n to a node m if there is an edge from n to m . If n is Mr. Odd's position, then the move is made by him, *i.e.* he should choose an out-going edge of n and move the token to the target node of the edge; Otherwise the move is made by Ms. Even. The result of a play is an infinite path $p : n_0 \rightarrow n_1 \rightarrow \dots$ starting from a given initial node n_0 . We say a play p is *dominated by an odd/even number* if $\max(\{D(n) \mid n \text{ occurs infinitely often in } p\})$ is odd/even. Mr. Odd wins the play if p is dominated by an odd number; Otherwise Ms. Even wins. A position is a *winning position* of a player if the player can win every play starting from the position regardless how his/her opponent moves. Parity games are *determinate* in the sense that every node in a game graph is a winning position of either Mr. Odd or Ms. Even.

In this talk, I will show that the set of the winning positions of each player in a game graph can be stratified into a hierarchy of *regions* and *spheres*. A simplest region is a cycle. A cycle is odd if the node with the largest weight in it is odd, and is even otherwise. In general a region R is a strongly connected component in the game graph consisting of cycles of the same polarity, *i.e.* the cycles in R are either all odd or all even. In the former case R is an odd-region, and in the later case it is an even-region. We shall call a node n an *inner node* of a region R if n is in R and all its out-going edges are also in R , and n a *border node* of R if n is in R and one out-going edge of n is not in R (thus a border node has exactly two out-going edges, one being in the region while the other not). An odd-region is *closed* if the source node of every edge going out of the region is a $\vee$ -node. An even-region is *closed* if the source node of every edge going out

^{*} Supported by National Natural Science Foundation of China (Grant No. 60721061).

of the region is a $\wedge$ -node. Thus, any $\wedge$ -node in a closed odd-region must be an inner node of the region, and any $\vee$ -node in a closed even-region must be an inner node of the region.

Intuitively, at any node in a closed odd-region R , Mr. Odd can steer a play from the node in such a way that the play will always stay within R . To this end he can use the following strategy, referred to as “odd-strategy” in the sequel. Recall that Mr. Odd’s positions are $\vee$ -nodes. If his current position n is an inner node of R , he may move the token along any out-going edge of n , and the next position will be in R ; If n is a border node of R then he moves the token along the out-going edge of n which is in R , and the next position will also be in R . On the other hand, since R is $\wedge$ -closed, every position of Ms. Even in R is an inner node of R , hence whatever edge she chooses to move the token the next position will still be inside R . Therefore if Mr. Odd adopts the odd-strategy the play will be entirely contained in R . Since every cycle in R is dominated by an odd node, the play, which is an infinite path consists of nodes and edges in R , will be dominated by some odd node in R , which means it is won by Mr. Odd. Therefore every node in a closed odd-region is Mr. Odd’s winning position. Dually, every node in a close even-region is Ms. Even’s winning position.

But not all nodes in any game graph fall into closed odd- or even-regions. To investigate the behaviour of plays starting from nodes outside closed regions, we need to introduce another notion. A *fan* is an acyclic subgraph with a root such that every node in the subgraph is reachable from the root. Since a fan is acyclic, it has some *end-nodes* all whose out-going edges are outside the fan. A node is an inner node of a fan if all its out-going edges are in the fan. A node is a border node of a fan if it has two out-going edges one of which is in the fan while the other is not. Thus a border node of a fan cannot be an end-node of the fan. A fan F is $\wedge$ -closed if the out-going edges of any inner $\wedge$ -node of F are inside F . In other words, every border node of F must be a $\vee$ -node. A node is in the *odd-sphere* of a set of odd-regions if it is either in some odd-region of the set, or is the root of a $\wedge$ -closed fan every whose end-node falls in some odd-region in the set. Let us call closed odd-regions in a game graph rank-0 odd-regions, and the odd-sphere of the set of rank-0 odd-regions the rank-0 odd-sphere of the graph. The notions of the rank-0 even-regions and the rank-0 even-sphere can be defined dually.

Let n be a node in the rank-0 odd-sphere. If n is in a closed odd-region then we already know it is Mr. Odd’s winning position. Otherwise n is the root of a $\wedge$ -closed fan. Then, using the odd-strategy as described above, Mr. Odd can steer any play starting from n so that, after a finite number of moves, the play will enter into some closed odd-region. From then on he can control the play to stay within that odd-region. Since only nodes which occurs infinitely often in a play are taken into account when deciding the winner of the play, the finite prefix of the play before entering the region does not affect the outcome of the play. Thus the play is also won by Mr. Odd. Therefore every node in the rank-0 odd-sphere of a game graph is Mr. Odd’s winning position. Dually, every node in the rank-0 even-sphere of a game graph is Ms. Even’s winning position.

To further extend the territory of regions and spheres, let us call an odd-region a *rank- k odd-region*, where $k > 0$, if every $\wedge$ -node lying on its border is the root of a $\wedge$ -closed fan whose end-points are in rank less than k odd-regions. The rank- k odd-sphere consists of the nodes in rank less than or equals to k odd-regions, plus those which are roots of $\wedge$ -closed fans whose end-nodes fall in rank less than or equals to k odd-regions. Thus, for any $k' < k$, the rank- k' odd-sphere is contained in the rank- k odd-sphere, but a rank- k' odd-region and a rank- k odd-region do not intersect. Since a game graph is finite, the hierarchy of ranked odd-regions/spheres is also finite, and we let the highest ranked odd-sphere be the odd-sphere of the graph.

From a game-theoretical point of view, for any play starting from a node n in the odd-sphere, if Mr. Odd applies the odd-strategy then, after a finite (could be zero) number of moves, the play will enter into a rank- k odd-region R for some k . At every $\vee$ -node of R Mr. Odd can manage to let the next position inside R . Only at a $\wedge$ -node m which lies on the border of R , it is possible for Ms. Even to move along an out-edge of m going out of R . However, since m is the root of a $\wedge$ -closed odd-fan whose end-nodes are in rank less than k odd-regions, Mr. Odd can control the play to arrive at an odd-region of rank- k' odd-region for some $k' < k$. Since the least rank is 0, in the end the play will stay inside some odd-region (which will be of rank-0 if Ms. Even has done her best) forever. Thus the play will be won by Mr. Odd. Therefore every node in the odd-sphere is a winning position of Mr. Odd.

Dually, we can define ranked even-regions, ranked even-spheres, and the even-sphere of a game graph. By duality, every node in the even-sphere is a winning position of Ms. Even. Furthermore, the odd-sphere and even-sphere do not intersect, and together they cover all the nodes in the graph. Thus the odd-sphere corresponds exactly to the set of winning positions of Mr. Odd, and the even-sphere corresponds exactly to the set of winning positions of Ms. Even. By this correspondence, the winning positions in parity games can be stratified into the hierarchies of ranked odd-/even-regions/spheres, as demonstrated above.

On the Physical Basics of Information Flow

- Results Obtained in Cooperation with Konrad Zuse -

Carl Adam Petri

St. Augustin, Germany

ca-petri@t-online.de

<http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri.html>

For three years Konrad Zuse (1910 - 1995) and me, we collaborated on the idea of a Computing Universe. We both agreed that some of the main tenets of modern physics would have to be expressed, at least those of quantum mechanics and of special relativity. Discussing which tenets should be tackled, Zuse said “those which can be understood by an engineer”. But many years passed before the deterministic approach of Gerard 't Hooft (2002) made a complete elaboration of the originally conceived ideas possible. We follow the principles of combinatorial modelling, which is a proper synthesis of continuous and discrete modelling.

A revision of the order axioms of measurement turns out to incorporate the uncertainty principle in a natural way. Besides this revision, the main innovations are a synthesis of the notions “discrete” and “continuous” and the derivation of computing primitives from smallest closed signal spaces. By means of NET modelling, we translate the main tenets of modern physics into their combinatorial form. In that form they are independent of scale, and relate to direct experience as well as to the sub-microscopic level of quantum foam.

Measurement, in the classical sense, is related to the uncertainty principle. While determinism excludes the creation of information, we go one tentative step further and forbid the destruction of information, in order to establish a law of conservation of information as a prototype of conservation laws in general. Accordingly, we describe the physical universe in terms of Signal Flow and, equivalently, of Information Flow. We derive the information operators from the idea of space-time periodic movement of signals in an integer Minkowski space. The derived loss free computing primitives have the same topology as the simplest patterns of repetitive group behaviour.

We can fulfil several systematic principles of construction in one single step. Each of those principles alone leads to the same result: the construction of loss-free TRANSFERS, which permits, in the long view, a great simplification. It follows that, if we base our models on the combinatorial concepts of signal flow suggested by informatics, and insist on continuity (as Zuse did), we end up inevitably with a model of a finite universe.

Faster Unfolding of General Petri Nets Based on Token Flows

Robin Bergenthum, Robert Lorenz, and Sebastian Mauser*

Department of Applied Computer Science,
Catholic University of Eichstätt-Ingolstadt,
firstname.lastname@ku-eichstaett.de

Abstract. In this paper we propose two new unfolding semantics for general Petri nets combining the concept of prime event structures with the idea of token flows developed in [11]. In contrast to the standard unfolding based on branching processes, one of the presented unfolding models avoids to represent isomorphic processes while the other additionally reduces the number of (possibly non-isomorphic) processes with isomorphic underlying runs. We show that both proposed unfolding models still represent the complete partial order behavior. We develop a construction algorithm for both unfolding models and present experimental results. These results show that the new unfolding models are much smaller and can be constructed significantly faster than the standard unfolding.

1 Introduction

Non-sequential Petri net semantics can be classified into unfolding semantics, process semantics, step semantics and algebraic semantics [17]. While the last three semantics do not provide semantics of a net as a whole, but specify only single, deterministic computations, unfolding models are a popular approach to describe the complete behavior of nets accounting for the fine interplay between concurrency and nondeterminism.

To study the behavior of Petri nets primarily two models for unfolding semantics were retained: labeled occurrence nets and event structures. In this paper we consider general Petri nets, also called place/transition Petri nets or p/t-nets (Figure 1). The standard unfolding semantics for p/t-nets is based on the developments in [19,5] (see [14] for an overview) in terms of so called branching processes, which are acyclic occurrence nets having events representing transition occurrences and conditions representing tokens in places. Branching processes allow events to be in conflict through branching conditions. Therefore branching processes can represent alternative processes simultaneously (processes are finite branching processes without conflict). Branching processes were originally introduced in [19] for safe nets,

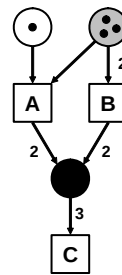


Fig. 1. Example net N. Instead of place-names we used different colors.

* Supported by the project SYNOPS of the German research council.

and extended in [5] to initially one marked p/t-nets without arc weights, by viewing the tokens as individualized entities. In contrast to [5], branching processes for p/t-nets even individualize tokens having the same "history", i.e. several (concurrent) tokens produced by some transition occurrence in the same place are distinguished through different conditions (see [14]). Analogously as in [5] one can define a single maximal branching process, called the unfolding of the system (in the rest of the paper we will refer to this unfolding as the standard unfolding). The unfolding includes all possible branching processes as prefixes, and thus captures the complete non-sequential branching behavior of the p/t-net. In the case of bounded nets, according to a construction by McMillan [16] a complete finite prefix of the unfolding preserving full information on reachable markings can always be constructed. This construction was generalized in [3] to unbounded nets through building equivalence classes of reachable markings. In the case of bounded nets, the construction of unfoldings and complete finite prefixes is well analyzed and several algorithmic improvements are proposed in literature [7,15,13]. By restricting the relations of causality and conflict of a branching process to events, one obtains a labeled prime event structure [20] underlying the branching process, which represents the causality between events of the branching process. An event structure underlying a process, i.e. without conflict, is called a run. In the view of the development of fast model-checking algorithms employing unfoldings resp. event structures [6] there is still the important problem of efficiently building them.

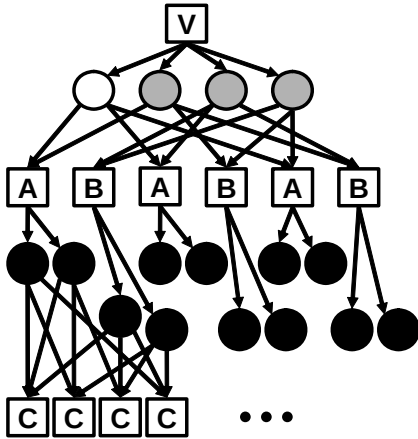


Fig. 2. Standard unfolding of N . The colors of the conditions refer to the place the corresponding tokens belong to.

the same "history" (for each place label), where two conditions have the same "history" if they have the same pre-event. Such events are called *strong identical*. In Figure 2 all A -labeled and all B -labeled events are strong identical, since all grey conditions have the same "history". Strong identical events produce isomorphic processes in the unfolding and therefore are redundant.

The p/t-net shown in Figure 1 has a finite standard unfolding (as defined for example in [14]). A part of this unfolding is shown in Figure 2. An unfolding has a unique minimal event producing the initial conditions. Each condition in the unfolding corresponds to a token in the net, i.e. tokens are individualized. In the initial marking there are three possibilities for transition B to consume two tokens from the grey place (and for transition A to consume one token from the grey and one token from the white place). All these possibilities define Goltz-Reisig processes [8] of the net, are in conflict and are reflected in the unfolding. That means, individualized tokens cause the unfolding to contain events with the same label, being in conflict and having the same number of equally labeled pre-conditions with

After the occurrence of transitions A and B there are four tokens in the black place and there are four possibilities for transition C to consume three of these tokens (Figure 2). For each of those possibilities, a C -labeled event is appended to the branching process. Two of these events consume two tokens produced by A and one token produced by B (these are strong identical), the other two consume one token produced by A and two tokens produced by B (these are also strong identical). The first pair of strong identical C -events is not strong identical to the second pair, but they all causally depend on the same two events. Such events having the same label, being in conflict and depending causally from the same events, are called *weak identical*. Weak identical events produce processes with isomorphic underlying runs and therefore also are redundant. Note finally, that the described four weak identical C -labeled events are appended to each of the three consistent pairs of A - and B -labeled events. That means, the individualized tokens in the worst case increase the number of events exponentially for every step of depth of the unfolding.

Figure 3 illustrates the labeled prime event structure underlying the unfolding shown in Figure 2. Formally a prime event structure is a partially ordered set of events (transition occurrences) together with a set of (so called) *consistency sets* [20]. "History-closed" (left-closed) consistency sets represent partially ordered runs. The labeled prime event structure underlying an unfolding is obtained by omitting the conditions and keeping the causality and conflict relations between events. Events not being in conflict define consistency sets. Thus, left-closed consistency sets correspond to processes and their underlying runs in the unfolding. Strong and weak identical events lead to consistency sets corresponding to processes with isomorphic underlying runs in the prime event structure.

In this paper we are interested in algorithms to efficiently construct unfoldings. As explained, the standard unfolding has the drawback, that it contains a lot of redundancy in form of isomorphic processes and processes with isomorphic underlying runs. This is caused by the individualization of tokens producing strong and weak identical events. Unfolding models with less nodes could significantly decrease the construction time, because a construction algorithm in some way always has to test all co-sets of events or conditions of the so-far constructed model to append further events. In this paper we propose two unfolding approaches reducing the number of events in contrast to the standard unfolding by neglecting (strong resp. weak) identical events.

Instead of considering branching processes, we use labeled prime event structures and assign so called token flows to its edges. Token flows were developed in [11] for a compact representation of processes. Token flows abstract from the individuality of conditions of a process and encode the flow relation of the process by natural numbers. For each place natural numbers are assigned to the edges of the partially ordered run

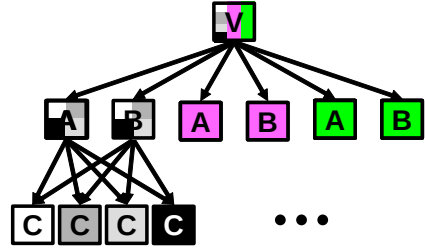


Fig. 3. Prime event structure of N . The colors of the events refer to the consistency sets. Transitive arcs are omitted.

underlying a process. Such a natural number assigned to an edge (e, e') represents the number of tokens produced by the transition occurrence e and consumed by the transition occurrence e' in the respective place. This principle is generalized to branching processes/unfoldings and their underlying prime event structures in this paper.

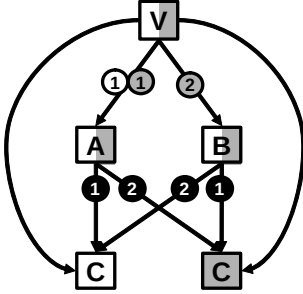


Fig. 4. Token flow unfolding of N . The coloring of the events illustrates the sets of consistent events.

The idea is to annotate each edge of the prime event structure underlying a branching process by the numbers of conditions between the corresponding pair of events of the branching process and omit isomorphic consistency sets having equal annotated token flow. The resulting prime event structure is shown in Figure 4. The event v is the unique initial event producing the initial marking. The edges have attached natural numbers, which are interpreted as token flows as described, where the colors refer to the places the tokens belong to. The assigned token flow specifies in particular that transition A consumes one initial token from the white place and one initial token from the grey place, while transition B consumes two initial tokens from the grey place. That means in this model the different possibilities for transition A and B of consuming initial tokens are not distinguished. Transition C either consumes one token produced by A and two tokens produced by B or vice versa in the black place. The respective two C -labeled events having the same pre-events but a different token flow are distinguished. They are in conflict yielding different consistency sets. In this approach strong identical events are avoided, while weak identical events still exist. Figure 4 only contains one of the three A and B events and two of the twelve C events. However, full information on reachable markings is still available. For example, the sum of all token flows assigned to edges from the initial event v to consistent events equals the initial marking. The example shows that through abstracting from the individuality of conditions, it is possible to generate an unfolding in form of a prime event structure with assigned token flow information having significantly less events than the standard unfolding.

A prime event structure with assigned token flow information is called a *token flow unfolding* if left-closed consistency sets represent processes and there are no strong identical events which are in conflict. Observe that to represent all possible processes we have to allow strong identical events which are not in conflict. For a given marked p/t-net, it is possible to define a unique maximal token flow unfolding, in which each process is represented through a consistency set with assigned token flows corresponding to the process. Figure 4 shows the maximal token flow unfolding for the example net N . We will show that the maximal token flow unfolding contains isomorphic processes only in specific situations involving auto-concurrency.

The token flow unfolding from Figure 4 still contains processes (consistency sets) which have isomorphic underlying runs, since token flow unfoldings still allow for weak identical events. In Figure 5 a prime event structure with assigned token flow

information is shown without weak identical events. Namely, the two weak identical C -labeled events in Figure 4 do not occur in Figure 5. This causes that the token flow information is not any more complete in contrast to token flow unfoldings, i.e. not each possible token flow distribution resp. process is represented. Instead example token flows are stored for each partially ordered run, i.e. each run is represented through one possible process. Note that still in this reduced unfolding full information on reachable markings is present, since markings reached through occurrence of a run do not depend on the token flow distribution within this run.

If a prime event structure with assigned token flow information does not contain weak identical events, this unfolding model is called a *reduced token flow unfolding*. We can define a unique maximal reduced token flow unfolding, in which each run is represented through a left-closed consistency set with assigned token flows corresponding to a process having this underlying run. It can be seen as a very compact unfolding model capturing the complete behavior of a p/t-net.

Figure 5 shows the maximal reduced token flow unfolding for the example net N . We will show that the maximal reduced token flow unfolding contains processes with isomorphic underlying runs only in specific situations involving auto-concurrency.

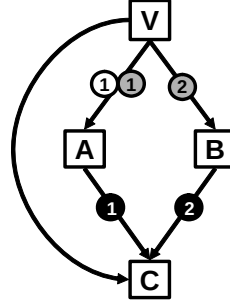


Fig. 5. Reduced token flow unfolding

For both new unfolding approaches we develop a construction algorithm for finite unfoldings and present an implementation together with experimental results. Token flow unfoldings can be constructed in a similar way as branching processes. The main difference is that processes are not implicitly given through events being in conflict, but are explicitly stored in consistency sets. This implies that new events are appended to consistency sets and not to co-sets of conditions. From the token flow information we can compute, how many tokens in which place, produced by some event, are still not consumed by subsequent events. These tokens can be used to append a new event. The crucial advantage of token flow unfoldings is that much less events must be appended. One disadvantage is that a possible exponential number of consistency sets must be stored. Moreover, for the construction of the reduced token flow unfolding not the full token flow information is available, since not each possible but only one example token flow distribution is displayed. Therefore the procedure of appending a new event is more complicated, because eventually an alternative token flow distribution has to be calculated (there is an efficient method for this calculation based on the ideas in [11]). Experimental results show that the two new unfolding models can be constructed much faster and memory consumption is decreased. The bigger the markings and arc weights are, the more efficient is the new construction compared to the standard one.

Altogether, the two new unfolding approaches on the one hand allow a much more efficient construction, and on the other hand still offer full information on concurrency, nondeterminism, causality and reachable markings. In particular, the assigned token

flows allow to compute the reachable marking corresponding to a consistency set. This allows to apply the theory of complete finite prefixes of the standard unfolding also to the presented new models. Acceleration of model checking algorithms working on the standard unfolding can be done by adapting them to the new smaller unfolding models. Another benefit is, that the new methods may lead to a more efficient computation of the set of all processes of a p/t-net.

There are also other attempts to extend the unfolding approach of [19] for safe nets to p/t-nets, where in some of them tokens are individualized as in the standard unfolding ([17,18]) and in some of them such an individualization of tokens is avoided as in our approach ([10,9,2,12,1]). In [17,18] conditions are grouped into families (yielding so called decorated occurrence nets) in order to establish desirable algebraic and order-theoretic properties. In [10] so called local event structures instead of prime event structures are introduced as an unfolding semantics of p/t-nets without autoconcurrency. In this approach, conflict and causality relations among events are not any more explicitly given by the model structure. Algorithmic aspects are not considered. In [2] arbitrarily valued and non-safe occurrence nets are used. Also here the direct link with prime event structures is lost. In [9], general nets are translated into safe nets by introducing places for reachable markings (which considerably increases the size of the structure) in order to apply branching processes for safe nets. In [1] a swapping equivalence between conditions introduces a collective token view.¹ Finally, in [12] events and conditions are merged according to appropriate rules yielding a more compact unfolding structure which not longer needs to be acyclic. Nevertheless it can be used for model checking. It cannot be directly computed from the p/t-net but only from its finite complete prefix which needs to be computed first. In contrast to all these approaches we propose a compact unfolding semantics avoiding individualized tokens while still explicitly reflecting causal relations between events through prime event structures. Moreover, basic order theoretic properties such as the existence of a unique maximal unfolding can be established. Our main focus is the development of fast construction algorithms, established so far for the finite case.

The remainder of the paper is organized as follows: In Section 2 we introduce basic mathematical notations and briefly restate the standard unfolding approach for p/t-nets. In Section 3 we develop the two new unfolding models. We prove that in both cases there is a unique maximal unfolding representing all processes resp. runs and formalize in which cases isomorphic processes resp. runs are avoided. Finally, in Section 4 we present algorithms for the construction of the new unfolding models in the finite case and provide experimental results in Section 5.

2 P/T-Nets and Standard Unfolding Semantics

In this section we recall the definitions of place/transition Petri nets and the standard unfolding semantics based on branching processes. We begin with some basic mathematical notations.

¹ Note here that the token flow unfolding and the reduced token flow unfolding define an equivalence on processes which is finer than the swapping equivalence, i.e. weak and strong equivalent events always produce swapping equivalent processes.

We use $\mathbb{N}$ to denote the *nonnegative integers*. A *multi-set* over a set A is a function $m : A \rightarrow \mathbb{N} \in \mathbb{N}^A$. For an element $a \in A$ the number $m(a)$ determines the number of occurrences of a in m . Given a binary relation $R \subseteq A \times A$ over a set A , the symbol R^+ denotes the *transitive closure* of R and R^* denotes the *reflexive transitive closure* of R . A *directed graph* G is a tuple $G = (V, \rightarrow)$, where V is its set of *nodes* and $\rightarrow \subseteq V \times V$ is a binary relation over V called its set of *arcs*. As usual, given a binary relation $\rightarrow$, we write $a \rightarrow b$ to denote $(a, b) \in \rightarrow$. For $v \in V$ and $W \subseteq V$ we denote by $\bullet v = \{v' \in V \mid v' \rightarrow v\}$ the *preset* of v , and by $v^\bullet = \{v' \in V \mid v \rightarrow v'\}$ the *postset* of v , $\bullet W = \bigcup_{w \in W} \bullet w$ is the *preset* of W and $W^\bullet = \bigcup_{w \in W} w^\bullet$ is the *postset* of W .

A *partial order* is a directed graph $(V, <)$, where $< \subseteq V \times V$ is an irreflexive and transitive binary relation. In this case, we also call $<$ a partial order. In the context of this paper, a partial order is interpreted as "earlier than"-relation between events. Two nodes (events) $v, v' \in V$ are called *independent* if $v \not< v'$ and $v' \not< v$. By $\text{co}_{<} \subseteq V \times V$ we denote the set of all pairs of independent nodes of V . A *co-set* is a subset $S \subseteq V$ fulfilling $\forall x, y \in S : x \text{ co}_{<} y$. A *cut* is a maximal co-set. For a co-set S and a node $v \in V \setminus S$ we write $v < S$ ($v > S$), if $\exists s \in S : v < s$ ($\exists s \in S : v > s$), and $v \text{ co}_{<} S$, if $\forall s \in S : v \text{ co}_{<} s$. A node v is called *maximal* if $v^\bullet = \emptyset$, and *minimal* if $\bullet v = \emptyset$. A subset $W \subseteq V$ is called *left-closed* if $\forall v, v' \in V : (v \in W \wedge v' < v) \implies v' \in W$. For a left-closed subset $W \subseteq V$, the partial order $(W, <|_{W \times W})$ is called *prefix* of $(V, <)$, defined by W . The *left-closure* of a subset W is given by the set $W \cup \{v \in V \mid \exists w \in W : v < w\}$. The node set of a finite prefix equals the left-closure of the set of its maximal nodes. Given two partial orders $\text{po}_1 = (V, <_1)$ and $\text{po}_2 = (V, <_2)$, we say that po_2 is a *sequentialization* of po_1 if $<_1 \subseteq <_2$. By $<_s \subseteq <$ we denote the smallest subset $<'$ of $<$ which fulfils $(<')^+ = <$, called the *skeleton* of $<$.

A *labeled partial order* (LPO) is a triple $(V, <, l)$, where $(V, <)$ is a partial order, and l is a *labeling function* on V . We use all notations defined for partial orders also for LPOs. If V is a set and $l : V \rightarrow X$ is a labeling function on V , then for a finite subset $W \subseteq V$, we define the multi-set $l(W) \subseteq \mathbb{N}^X$ by $l(W)(x) = |\{v \in W \mid l(v) = x\}|$. LPOs are used to represent partially ordered runs of Petri nets. Such runs are distinguished only up to isomorphism. Two LPOs $(V_1, <_1, l_1)$ and $(V_2, <_2, l_2)$ are *isomorphic* if there is a bijective mapping $\varphi : V_1 \rightarrow V_2$ satisfying $\forall v_1 \in V_1 : l_1(v_1) = l_2(\varphi(v_1))$ and $\forall v_1, v'_1 \in V_1 : v_1 <_1 v'_1 \iff \varphi(v_1) <_2 \varphi(v'_1)$.

A *net* is a triple $N = (P, T, F)$, where P is a set of *places*, T is a set of *transitions*, satisfying $P \cap T = \emptyset$, and $F \subseteq (P \cup T) \times (T \cup P)$ is a *flow relation*. Places and transitions are called the nodes of N . Presets and postsets of (sets of) nodes are defined w.r.t. the directed graph $(P \cup T, F)$. We denote $\preceq_N = F^*$ and $\prec_N = F^+$. If N is clear from the context, we also write $\preceq$ instead of $\preceq_N$ and $\prec$ instead of $\prec_N$.

Assume now that $\prec_N = \prec$ is a partial order. Then two nodes x, y (places or transitions) of N are *in conflict*, denoted by $x \# y$, if there are distinct transitions $t, t' \in E$ with $\bullet t \cap \bullet t' \neq \emptyset$ such that $t \preceq x$ and $t' \preceq y$. Two nodes x, y are called *independent* if $x \text{ co}_{\prec} y$ and $\neg(x \# y)$. Maximal and minimal nodes of N and prefixes of N are defined w.r.t. $(P \cup T, \prec)$.

Definition 1 (Place/transition net). A place/transition-net (shortly p/t-net) N is a quadruple (P, T, F, W) , where (P, T, F) is a net with finite sets of places and transitions, and $W : F \rightarrow \mathbb{N} \setminus \{0\}$ is a weight function. A marking of a p/t-net $N = (P, T, F, W)$

is a function $m : P \rightarrow \mathbb{N}$. A marked p/t-net is a pair (N, m_0) , where N is a p/t-net, and m_0 is a marking of N , called initial marking.

We extend the weight function W to pairs of net elements $(x, y) \in (P \times T) \cup (T \times P)$ satisfying $(x, y) \notin F$ by $W((x, y)) = 0$. A transition $t \in \mathbb{N}$ is *enabled to occur* in a marking m of N if $\forall p \in P : m(p) \geq W((p, t))$. If t is enabled to occur in a marking m , then its *occurrence* leads to the new marking m' defined by $m'(p) = m(p) - W((p, t)) + W((t, p))$ for $p \in P$.

Unfolding semantics of p/t-nets is given by so called *branching processes* which are based on occurrence nets. A conflict relation between events distinguishes alternative runs. Runs are given by conflict-free, left-closed sub-nets of branching processes.

Definition 2 (Occurrence net). An occurrence net is a net $O = (B, E, G)$ satisfying

- O is acyclic, i.e. $\prec_O$ is a partial order.
- $\forall b \in B : |\bullet b| \leq 1$.
- $\forall x \in B \cup E : \neg(x \# x)$.
- $\forall x \in B \cup E : |\{y \mid y \prec x\}|$ is finite.

Elements of B are called *conditions* and elements of E are called *events*. $MIN(O)$ denotes the set of minimal elements (w.r.t. $\prec_O$).

Definition 3 (Branching process). Let (N, m_0) , $N = (P, T, F, W)$ be a marked p/t-net. A branching process of (N, m_0) is a pair $\pi = (O, \rho)$ where $O = (B, E, G)$ is an occurrence net and $\rho : B \cup E \rightarrow X$ with $P \cup T \subset X$ is a labeling function satisfying:

- There is $e_{init} \in E$ with $MIN(O) = \{e_{init}\}$ and $\rho(e_{init}) \notin P \cup T$.
- $\forall b \in B : \rho(b) \in P$ and $\forall e \in E \setminus \{e_{init}\} : \rho(e) \in T$.
- $\forall e \in E \setminus \{e_{init}\}, \forall p \in P : |\{b \in \bullet e \mid \rho(b) = p\}| = W((p, \rho(e))) \wedge |\{b \in e^\bullet \mid \rho(b) = p\}| = W((\rho(e), p))$.
- $\forall p \in P : |\{b \in e_{init}^\bullet \mid \rho(b) = p\}| = m_0(p)$.
- $\forall e, f \in E : (\bullet e = \bullet f \wedge \rho(e) = \rho(f)) \implies (e = f)$.

In a branching process, $\prec$ is interpreted as "earlier than"-relation between transition occurrences. A finite branching process with empty conflict relation is called a *process*.

Two branching processes $\pi' = (O', \rho')$, $O' = (B', E', G')$, and $\pi = (O, \rho)$, $O = (B, E, G)$, are isomorphic, if there is a bijection $Iso : B \cup E \rightarrow B' \cup E'$ satisfying $Iso(B) = B'$, $Iso(E) = E'$, $\rho' \circ Iso = \rho$ and $(x, y) \in G \iff (Iso(x), Iso(y)) \in G'$ for $x, y \in B \cup E$.

A branching process $\pi = (O, \rho)$, $O = (B, E, G)$, is a prefix of another branching process $\pi' = (O', \rho')$, $O' = (B', E', G')$, denoted by $\pi \sqsubseteq \pi'$, if O is a prefix of O' satisfying $B = MIN(O) \cup (\bigcup_{e \in E} e^\bullet)$ and ρ is the restriction of ρ' to $B \cup E$. For each marked p/t-net (N, m_0) there exists a unique, w.r.t. $\sqsubseteq$ maximal, branching process $\pi_{max}(N, m_0)$, called the *unfolding* of (N, m_0) .

Sometimes one is only interested in storing the causal dependencies of events of a branching process. For this conditions are omitted and the $\prec$ - and $\#$ -relation are kept for events. Formally the resulting object is a so-called *prime event structure*.

Definition 4 (Prime event structure). A prime events structure is a triple $PES = (E, Con, \prec)$ consisting of a set E of events, a partial order $\prec$ on E and a set Con of finite subsets of E satisfying:

- $\forall e \in E : \{e' \mid e' \prec e\}$ is finite.
- $\{e\} \in Con$.
- $Y \subseteq X \in Con \implies Y \in Con$.
- $((X \in Con) \wedge (\exists e' \in X : e \prec e')) \implies (X \cup \{e\} \in Con)$.

A consistent subset of E is a subset X satisfying $\forall Y \subseteq X, Y$ finite : $Y \in Con$. The conflict relation $\#$ between events of PES is defined by $e \# e' \iff \{e, e'\} \notin Con$.

A pair (PES, l) , where PES is a prime events structure and l is a labeling function on E , is called labeled prime event structure.

A (labeled) prime event structure with E consistent we interpret as an LPO, i.e. in this case we omit the set of consistency sets Con .

If $\pi = (O, \rho)$, $O = (B, E, G)$ is a branching process, then $PES(\pi) = (E, Con, \prec \upharpoonright_{E \times E})$, where $X \in Con$ if and only if $X \subseteq E$ is finite and fulfills $\forall e, e' \in X : \neg(e \# e')$, is a prime event structure. $PES(\pi)$ is called *corresponding* to π . If π is a process, then $PES(\pi)$ is a finite LPO, called the *run underlying* π .

3 Unfoldings Based on Token Flows

One basic problem of the unfolding of a p/t-net is, that it contains a lot of redundancy. This arises from the individuality of conditions in branching processes. When appending a new transition occurrence to a branching process, each particular choice of a set of conditions representing the preset of this transition yields a different process, where some of these processes are isomorphic and others have isomorphic underlying runs (see Figure 2 and the explanations in the introduction). In this section we propose two new unfolding semantics of p/t-nets avoiding such redundancy. Both approaches are based on the notion of token flows presented in [11]. In the following we restate this notion and its role in the representation of single processes. In the next subsection, the concepts will be transferred to unfoldings.

In the following, LPOs are considered to be finite. The edges of LPOs, representing partially ordered runs of a p/t-net, are annotated by (tuples of) non-negative integers interpreted as token flow between transition occurrences. Namely, for a process $K = (O, \rho)$, $O = (B, E, G)$, of a marked p/t-net (N, m_0) , $N = (P, T, F, W)$, we defined a so called *canonical token flow function* $x_K : \prec \rightarrow \mathbb{N}^P$ assigned to the edges of the run $(E, \prec, \rho)$ underlying K via $x_K((e, e')) = \rho(e^\bullet \cap \bullet e')$. That means $x_K((e, e'))$ represents for each place the number of tokens which are produced by the occurrence of the transition $\rho(e)$ and then consumed by the occurrence of $\rho(e')$. Such a token flow function abstracts away from the individuality of conditions in a process and encodes the token flow by natural numbers for each place. It is easy to see that x_K satisfies:

- (IN): $\forall e \in E \setminus \{e_{init}\}, \forall p \in P : (\sum_{e' \prec e} x_K(e', e))(p) = W(p, \rho(e))$.
- (OUT): $\forall e \in E \setminus \{e_{init}\}, \forall p \in P : (\sum_{e \prec e'} x_K(e, e'))(p) \leq W(\rho(e), p)$.

- (INIT): $\forall p \in P : (\sum_{e_{init} \prec e'} x(e_{init}, e'))(p) \leq m_0(p)$.
- (MIN): $\forall (e, e') \in \prec_s : (\exists p \in P : x_K(e, e')(p) \geq 1)$.

(IN), (OUT) and (INIT) reflect the consistency of the token flow distribution given by x_K with the initial marking and the arc weights of the considered net. (MIN) holds since skeleton arcs define the "earlier than"-causality between transition occurrences and this causality is produced by non-zero token flow. Non-skeleton edges may carry a zero token flow, since they are induced by transitivity of the partial order. A zero flow of tokens means, that there is no direct dependency between events. In particular, there are no token flows between concurrent events.

In [11] we showed, that the other way round for an LPO $(E, \prec, \rho)$ with unique initial event e_{init} , a token flow function $x : \prec \rightarrow \mathbb{N}^P$ satisfying (IN), (OUT), (INIT) and (MIN) is a canonical token flow function of a process. That means processes are in one-to-one correspondence with LPOs having token flow assigned to their edges fulfilling (IN), (OUT), (INIT) and (MIN). Such LPOs yield an equivalent but more compact representation of partially ordered runs. In particular full information on reachable markings as well as causal dependency and concurrency among events is preserved.

3.1 Token Flow Unfolding

In the following, we extend these ideas to branching processes by assigning token flows to the edges of prime event structures. We will prove that in such a way one gets a more compact representation of the branching behavior of p/t-nets while preserving full information on markings, concurrency, causal dependency and conflict.

Let $PES = (E, Con, \prec)$ be a prime event structure. We denote the set of left-closed consistency sets by $Con_{pre} \subseteq Con$. For a *token flow function* $x : \prec \rightarrow \mathbb{N}^P$, a consistency set $C \in Con_{pre}$ and an event $e \in C$ we denote

- $IN_C(e) = \sum_{e' \prec e, e' \in C} x(e', e)$ the *intoken flow* of e w.r.t. C .
- $OUT_C(e) = \sum_{e \prec e', e' \in C} x(e, e')$ the *outtoken flow* of e w.r.t. C .

A prime token flow event structure is a labeled prime event structure together with a token flow function. Since equally labeled events represent different occurrences of the same transition, they are required to have equal intoken flow. Since not all tokens which are produced by an event are consumed by further events, there is no analogous requirement for the outtoken flow. It is assumed that there is a unique initial event producing the initial marking.

Definition 5 (Prime token flow event structure). A prime token flow event structure is a pair $((PES, l), x)$, where $PES = (E, Con, \prec)$ is a prime event structure, l is a labeling function on E and $x : \prec \rightarrow \mathbb{N}^P$ is a token flow function satisfying:

- There is a unique minimal event e_{init} w.r.t. $\prec$ with $l(e_{init}) \neq l(e)$ for all $e \neq e_{init}$.
- $\forall C, C' \in Con_{pre}, \forall e \in C, e' \in C' : l(e) = l(e') \implies IN_C(e) = IN_{C'}(e')$.

A token flow unfolding of a marked p/t-net is a prime token flow event structure, in which intoken and outtoken flows are consistent with the arc weights resp. the initial

marking of the net within each left-closed consistency set. Moreover, we neglect so-called *strong identical events* in such unfoldings which turn out to produce isomorphic process nets.²

Definition 6 (Strong identical events). Let $((PES, l), x)$ be a prime token flow event structure. Two events $e, e' \in E$ fulfilling

$$(l(e) = l(e')) \wedge (\bullet e = \bullet e') \wedge (\forall f \in \bullet e : x(f, e) = x(f, e'))$$

are called strong identical.

Definition 7 (Token flow unfolding). Let (N, m_0) , $N = (P, T, F, W)$, be a marked p/t-net. A token flow unfolding of (N, m_0) is a prime token flow event structure $((PES, l), x)$, $l : E \rightarrow X$ with $T \subset X$ and $\forall e \in E \setminus \{e_{init}\} : l(e) \in T$, satisfying:

- (IN): $\forall C \in Con_{pre}, \forall e \in C \setminus \{e_{init}\}, \forall p \in P : IN_C(e)(p) = W(p, l(e))$.
- (OUT): $\forall C \in Con_{pre}, \forall e \in C \setminus \{e_{init}\}, \forall p \in P : OUT_C(e)(p) \leq W(l(e), p)$.
- (INIT): $\forall C \in Con_{pre}, \forall p \in P : OUT_C(e_{init})(p) \leq m_0(p)$.
- (MIN): $\forall (e, e') \in \prec_s : (\exists p \in P : x(e, e')(p) \geq 1)$.
- There are no strong identical events e, e' satisfying $\{e, e'\} \notin Con$.

Two token flow unfoldings $\mu' = ((PES', l'), x')$, $PES' = (E', Con', \prec')$, and $\mu = ((PES, l), x)$, $PES = (E, Con, \prec)$, are isomorphic if there is a bijection $I : E \rightarrow E'$ satisfying $\forall e \in E : l(e) = l'(I(e)) \wedge I(\bullet e) = \bullet I(e) \wedge I(e \bullet) = I(e) \bullet$, $\forall C \subseteq E : C \in Con \Leftrightarrow I(C) \in Con'$ and $\forall e \prec f : x(e, f) = x'(I(e), I(f))$.

Given a token flow unfolding $\mu = ((PES, l), x)$, $PES = (E, Con, \prec)$, each non-empty left-closed subset $E' \subseteq E$ defines a token flow unfolding $\mu' = ((PES', l'), x')$, $PES' = (E', Con', \prec')$ by $Con' = \{C \in Con \mid C \subseteq E'\}$, $\prec' = \prec|_{E' \times E'}$, $l' = l|_{E'}$ and $x' = x|_{\prec'}$. Each token flow unfolding $\mu'' = ((PES'', l''), x'')$, $PES'' = (E', Con'', \prec'')$, $Con'' \subseteq Con'$ is called prefix of μ , denoted by $\mu'' \sqsubseteq \mu$. By $\sqsubseteq = \sqsubseteq \setminus id$, a partial order on the set of token flow unfoldings is given. We now want to prove that up to isomorphism, there exists a unique maximal (in general infinite) token flow unfolding w.r.t. $\sqsubseteq$. The partial order $\sqsubseteq$ can be defined through appending new events to (consistency sets of) existing token flow unfoldings starting with the initial event. There is a unique maximal token flow unfolding (fix point) only if the order of appending events does not matter, i.e. if events are appended in different orders, isomorphic token flow unfoldings are constructed.

A new transition occurrence can be appended to a consistency set, if there are enough remaining tokens produced by events in the consistency set (tokens which are not consumed by subsequent events in the consistency set). For $e \in E \setminus \{e_{init}\}$ and $C \in Con_{pre}$, the remaining tokens produced by e are formally given by the *residual token flow* $Res_C(e)$ (of e w.r.t. C) defined by $Res_C(e)(p) = W(l(e), p) - \sum_{e \prec e', e' \in C} x(e, e')(p)$ for $p \in P$. Similarly, for each $p \in P$, $Res_C(e_{init})(p) =$

² Note that omitting strong identical events disables the possibility of applying prime event structures having a set of consistency sets defined by a binary conflict relation, which is the case for example for prime event structures corresponding to branching processes.

$m_0(p) - \sum_{e_{init} \prec e', e' \in C} x(e_{init}, e')(p)$. Let $Mar(C) = \sum_{e \in C} Res_C(e)$ be the residual marking of C . If there are enough tokens in the residual marking to fire a transition t , there may be several choices which of the remaining tokens are used to append a respective transition occurrence to the consistency set. Each such choice is formally represented by an *enabling function* $y : C \rightarrow \mathbb{N}^P$ satisfying $\forall e \in C : Res_C(e) \geq y(e)$ and $\forall p \in P : (\sum_{e \in C} y(e))(p) = W((p, t))$. Such an enabling function defines a new event e_y by $l'(e_y) = t$, $\bullet e_y = \{e \in C \mid \exists e' : y(e') \neq 0 \wedge (e = e' \vee e \prec e')\}$ and $\forall e \in \bullet e_y : x'(e, e_y) = y(e)$. If there is already a strong identical event not belonging to the consistency set, then this strong identical event is added to the consistency set. Otherwise, the new event is added.

Definition 8 (Appending events). Let (N, m_0) , $N = (P, T, F, W)$, be a marked p/t -net and let $\mu = ((PES, l), x)$, $PES = (E, Con, \prec)$, be a token flow unfolding of (N, m_0) .

Let $t \in T$ and $C \in Con_{pre}$ be such that $Mar(C)(p) \geq W((p, t))$ for each p . Let y be an enabling function and e_y be the associated new event.

If there is no event $e \notin C$ which is strong identical to e_y , then we define a prime token flow event structure $Ext(\mu, C, y, t) = ((PES', l'), x')$, $PES' = (E', Con', \prec')$, through $E' = E \cup \{e_y\}$, $l'|_E = l$, $x'|_x = x$, $\prec'|_{E \times E} = \prec$ and $Con' = Con \cup \{C' \cup \{e_y\} \mid C' \subseteq C\}$. We say that μ is extended by e_y .

If there is an event $e_{id} \notin C$ which is strong identical to e_y , then we define a prime token flow event structure $Ext(\mu, C, y, t) = ((PES', l'), x')$, $PES' = (E', Con', \prec')$, through $E' = E$, $l' = l$, $x' = x$, $\prec' = \prec$ and $Con' = Con \cup \{C' \cup \{e_{id}\} \mid C' \subseteq C\}$. We say that μ is updated by e_y .

The following lemma ensures that we have defined an appropriate procedure to append events:

Lemma 1. $Ext(\mu, C, y, t)$ fulfills:

- (i) $Ext(\mu, C, y, t)$ is a token flow unfolding.
- (ii) $\mu \sqsubseteq Ext(\mu, C, y, t)$.
- (iii) Every finite token flow unfolding μ can be constructed by the procedure shown in Definition 8: Given a token flow unfolding $\mu = ((PES, l), x)$, $PES = (E, Con, \prec)$, there exists a sequence $\mu_0, \dots, \mu_n$ of token flow unfoldings with $\mu_0 = (((\{e_{init}\}, \{\{e_{init}\}\}, \emptyset), id), \emptyset)$, $\mu_n = \mu$ and $\mu_{i+1} = Ext(\mu_i, C_i, y_i, t_i)$ for $i = 0, \dots, n-1$.

Proof. The first and second statement follow by construction. The third one can be shown as follows: Fix one ordering of $E = \{e_1, \dots, e_n\}$, such that $e_i \prec e_j \implies i < j$ and denote $E_i = E \setminus \{e_i, \dots, e_n\}$. Then $\mu_0 \sqsubset \mu_{E_2} \sqsubset \dots \sqsubset \mu_{E_n} \sqsubset \mu_E$, where $\mu_{E'}$ is the prefix of μ defined by a left-closed set $E' \subseteq E$. By definition, there are triples $(C_1^i, y_1^i, l(e_i)), \dots, (C_m^i, y_m^i, l(e_i))$, such that $\mu_{E_{i+1}}$ can be constructed from μ_{E_i} by appending $l(e_i)$ -occurrences in arbitrary order to C_j^i via y_j^i for $j = 1, \dots, m$. Namely, $C_1^i, \dots, C_m^i$ are the sets arising by omitting e_i from every left-closed consistency set in $\mu_{E_{i+1}}$ which includes e_i and all y_j^i are defined according to the intoken flow of e_i . That means $supp_i = \{y_k^i > 0\} = \{y_j^i > 0\}$ and $y_k^i|_{supp_i} = y_j^i|_{supp_i}$ for all k, j . Therefore,

actually in the first appending step the event e_i is appended and in the further $m - 1$ steps only consistency sets are updated.

In the construction of the above proof, the resulting token flow unfolding does not depend on the used ordering of the events in E and also does not depend on the used ordering of the consistency sets enabling a fixed event e . This means that extending finite token flow unfoldings by new events in different orders and w.r.t. different consistency sets leads to isomorphic token flow unfoldings if after each extension all consistency sets which enable the considered event are updated. Observe moreover that by definition also the extension by a new event and the update by another event can be mixed up. This gives the following statement:

Lemma 2. *Let (N, m_0) be a marked p/t-net. There is a token flow unfolding $Unf_{max}(N, m_0)$, which is maximal w.r.t. $\sqsubset$ (no more events can be appended to finite prefixes) and unique up to isomorphism.*

$Unf_{max}(N, m_0)$ can be defined as the limit of a sequence of finite token flow unfoldings $(\mu_n)_{n \in \mathbb{N}}$ with $\mu_{n+1} = Ext(\mu_n, C, y, t)$, since the order of appending events does not matter. Each finite left-closed consistency set C of $Unf_{max}(N, m_0)$ represents a process π_C of (N, m_0) in the sense that the LPO $lpo_C = (C, \prec|_{C \times C}, l|_C)$ is the run underlying π_C and $x_C = x|_{C \times C}$ is the canonical token flow function of π_C (this follows from [11] since x_C satisfies (INIT), (IN), (OUT) and (MIN) on lpo_C). Moreover, in $Unf_{max}(N, m_0)$ all processes of (N, m_0) are represented by finite left-closed consistency sets. Namely, for each process, the underlying run with assigned canonical token flow defines a token flow unfolding and without loss of generality we can assume that this token flow unfolding equals μ_k of a defining sequence of $Unf_{max}(N, m_0)$ for some k .

Theorem 1. *Let (N, m_0) be a marked p/t-net. Then for each process π of (N, m_0) there is a left-closed consistency set C of $Unf_{max}(N, m_0)$ such that π_C is isomorphic to π .*

To show that $Unf_{max}(N, m_0)$ avoids the generation of isomorphic processes, we prove that only in special auto-concurrency situations processes of (N, m_0) are represented more than once in $Unf_{max}(N, m_0)$.

Theorem 2. *Let (N, m_0) be a marked p/t-net and π be a finite process of (N, m_0) . If in $Unf_{max}(N, m_0) = ((PES, l), x)$, $PES = (E, Con, \prec)$, there are two finite sets $C \neq C' \in Con_{pre}$ representing π , then there exist events $e \in C, e' \in C', e \neq e'$ such that e and e' are strong identical and fulfill $\{e, e'\} \in Con$.*

Proof. Assume there are finite $C, C' \in Con_{pre}$ such that the processes π_C and $\pi_{C'}$ are isomorphic. Let $e \in C \setminus C'$ with $\bullet e \subseteq C \cap C'$. Such an event e exists since $C \cap C'$ defines a prefix of PES containing e_{init} and therefore e can be chosen as a minimal element w.r.t. $\prec$ in $C \setminus C'$. Let $e' \in C'$ be the image of e under the isomorphism relating π_C and $\pi_{C'}$. Since π_C and $\pi_{C'}$ are isomorphic, the left-closed consistency sets of all pre-events of e resp. e' define isomorphic processes. Thus, either e and e' are strong identical, or there are $f \in \bullet e \setminus \bullet e'$ and $f' \in \bullet e' \setminus \bullet e$ fulfilling the same property

as e and e' that the left-closed consistency sets of all pre-events of f resp. f' define isomorphic processes. Since the number of pre-events of f and f' is smaller than the number of pre-events of e and e' (i.e. the procedure can only finitely often be iterated), and e_{init} is a common pre-event, there is some pair of events g and g' being strong identical. By definition we have $g \in C$, $g' \in C'$ and $g \neq g'$. The definition of token flow unfoldings ensures $\{g, g'\} \in Con$ (since g, g' are strong identical).

Since e and e' are strong identical, they in particular have the same label and $\{e, e'\} \in Con$ shows that they can occur concurrently in some marking.

Corollary 1. *If (N, m_0) allows no auto-concurrency (i.e. there is no reachable marking m , such that there is a transition t fulfilling $\forall p \in P : m(p) \geq 2 \cdot W((p, t))$), there is a one-to-one correspondence between left-closed consistency sets of $Unf_{max}(N, m_0)$ and (isomorphism classes of) processes.*

Although we have seen that the non-existence of strong identical events is not enough to avoid isomorphic processes in general, the number of isomorphic processes represented in token flow unfoldings is significantly smaller than in the standard unfolding approach (see the experimental results). Figure 6 shows a situation as discussed in Theorem 2, where the token flow unfolding as introduced so far is not small enough to completely neglect isomorphic processes. Namely, the two B -labeled events produce two isomorphic processes, despite they are not strong identical (because they have no common pre-events). Observe however, that the two A -labeled pre-events of the two B -labeled events are themselves strong identical, but are not in conflict (they are concurrent). To avoid such situation, we must generalize the notion of strong identical events in the sense that two strong identical events have not necessarily common, but strong identical pre-events. This setting is formally described by the notion of isomorphic strong identical events as follows:

Let $((PES, l), x)$ be a prime token flow event structure. Let $\cong \subseteq E \times E$ be the least equivalence relation satisfying for all $e, e' \in E$:

- $((l(e) = l(e')) \wedge (\bullet e = \bullet e') \wedge (\forall f \in \bullet : x(f, e) = x(f, e'))) \implies (e \cong e')$.
- $((l(e) = l(e')) \wedge (\exists I : \bullet e \rightarrow \bullet e' \text{ bijective} : (\forall f \in \bullet e : f \cong I(f) \wedge x(f, e) = x(I(f), e')))) \implies (e \cong e')$.

Two $\cong$ -equivalent events e, e' are called *isomorphic strong identical*. Basically, omitting isomorphic strong identical events yields a token flow unfolding representing no isomorphic (maximal) processes at all (can be deduced similarly as Theorem 2). But considering such an approach we encountered several intricate technical problems. In particular, a test for the isomorphic strong identical property is complicated, such that the algorithmic applicability is questionable.

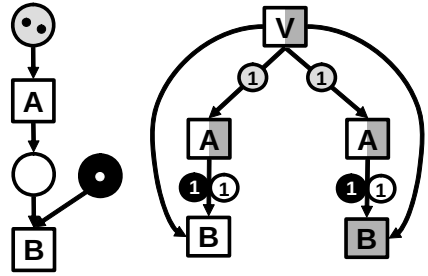


Fig. 6. P/t-net with token flow unfolding containing two isomorphic (maximal) processes

3.2 Reduced Token Flow Unfolding

In a token flow unfolding there is still redundancy w.r.t causality and concurrency, since there are consistency sets which induce processes which have the same underlying run (but a different token flow distribution). Such consistency sets are caused by so-called *weak identical events* (compare the introduction). To avoid weak identical events, since many different processes produce one run and token flow distributions correspond to processes, we store for each consistency set an example token flow distribution.

That means, we need to extend our model of prime event structures $PES = (E, Con, \prec)$ extended by token flows such that we can store for each consistency set $C \in Con_{pre}$ an individual token flow $x_C : \prec|_{C \times C} \rightarrow \mathbb{N}^P$. For such x_C and an event $e \in C$ we denote $IN_C(e) = \sum_{e' \prec_e} x_C(e', e)$ and $OUT_C(e) = \sum_{e \prec_{e'}} x_C(e, e')$. We introduce *generalized prime token flow event structures* as pairs $((PES, l), (x_C)_{C \in Con_{pre}})$, where $PES = (E, Con, \prec)$ is a prime event structure, l is a labeling function on E and $(x_C)_{C \in Con_{pre}}$ is a family of *token flow functions* $x_C : \prec|_{C \times C} \rightarrow \mathbb{N}^P$ satisfying analogous conditions as prime token flow event structures:

- There is a unique minimal event e_{init} w.r.t. $\prec$ with $l(e_{init}) \neq l(e)$ for all $e \neq e_{init}$.
- $\forall C, C' \in Con_{pre}, \forall e \in C, e' \in C' : l(e) = l(e') \implies IN_C(e) = IN_{C'}(e')$.

Two distinct events $e \in C, e' \in C'$, fulfilling $l(e) = l(e') \wedge \bullet e = \bullet e'$, are called *weak identical*.

Definition 9 (Reduced token flow unfolding). Let (N, m_0) , $N = (P, T, F, W)$, be a marked p/t-net. A reduced token flow unfolding of (N, m_0) is a generalized token flow unfolding $((PES, l), (x_C)_{C \in Con_{pre}})$ satisfying (IN), (OUT), (INIT),

$$(MIN): \forall C \in Con_{pre}, \forall e, e' \in C, e \prec_s e' : (\exists p \in P : x_C(e, e')(p) \geq 1)$$

and having no weak identical events e, e' satisfying $\{e, e'\} \notin Con$.

Similar as for token flow unfoldings, prefixes can be defined. Given a reduced token flow unfolding $\mu = ((PES, l), (x_C)_{C \in Con_{pre}})$, $PES = (E, Con, \prec)$, each non-empty left-closed subset $E' \subseteq E$ defines a reduced token flow unfolding $\mu' = ((PES', l'), (x'_C)_{C \in Con'_{pre}})$, $PES' = (E', Con', \prec')$ by $Con' = \{C \in Con \mid C \subseteq E'\}$, $\prec' = \prec|_{E' \times E'}$, $l' = l|_{E'}$ and $x'_C = x_C$. Each token flow unfolding $\mu'' = ((PES'', l''), (x'')_{C \in Con''_{pre}})$, $PES'' = (E', Con'', \prec'')$, $Con'' \subseteq Con'$ is called *prefix* of μ , denoted by $\mu'' \sqsubseteq \mu$.

Events can be appended to reduced token flow unfoldings similar as to token flow unfoldings. The residual token flow $Res_C(e)$ and the residual marking $Mar(C)$ are defined analogously as before, using x_C instead of x . If there are enough tokens in the residual marking to fire a transition t , in general a new token flow distribution for the considered consistency set has to be stored in order to have the possibility to append a respective transition occurrence to the consistency set via an enabling function y . Formally, such a token flow redistribution is given by a redistribution flow function $x : \prec|_{C \times C} \rightarrow \mathbb{N}^P$ fulfilling (IN), (OUT), (INIT) and $\forall (e, e') \in \prec_s \cap C \times C : (\exists p \in P : x(e, e')(p) > 0)$ such that there is an enabling function $y : C \rightarrow \mathbb{N}^P$ satisfying $\forall e \in C : W(l(e), p) - \sum_{e \prec_{e'}} x(e, e')(p) \geq y(e)$ and $\forall p \in P : (\sum_{e \in C} y(e))(p) = W((p, t))$. The functions x and y define a new event $e_{x,y}$ through $l'(e_{x,y}) = t$, $\bullet e_{x,y} =$

$\{e \in C \mid \exists e' : y(e') \neq 0 \wedge (e = e' \vee e \prec e')\}$ and $\forall e \in \bullet_{e_{x,y}} : x'(e, e_{x,y}) = y(e)$. If there is already a weak identical event not belonging to the consistency set, then this weak identical event is added to the consistency set. Otherwise, the new event is added.

Definition 10 (Appending events). *Let (N, m_0) , $N = (P, T, F, W)$, be a marked p/t-net and let $\mu = ((PES, l), (x_C)_{C \in Con_{pre}})$, $PES = (E, Con, \prec)$, be a reduced token flow unfolding of (N, m_0) .*

Let $t \in T$ and $C \in Con_{pre}$, such that $Mar(C)(p) \geq W((p, t))$ for each p . Let x be a redistribution function with associated enabling function y and $e_{x,y}$ be the corresponding new event.

If there is no event $e \notin C$ which is weak identical to $e_{x,y}$, then we define a generalized prime token flow event structure $Ext(\mu, C, x, y, t) = ((PES', l'), x')$, $PES' = (E', Con', \prec')$, through $E' = E \cup \{e_{x,y}\}$, $l'|_E = l$, $Con' = Con \cup \{C' \cup \{e_{x,y}\} \mid C' \subseteq C\}$, $\forall C' \in Con'_{pre}$, $e_{x,y} \in C' : x'_{C'}|_{\prec} = x \wedge x'_{C'}|_{C' \times \{e_{x,y}\}} = x'$ and $\prec'|_{E \times E} = \prec$. We say that μ is extended by $e_{x,y}$.

If there is an event $e_{id} \notin C$ which is weak identical to $e_{x,y}$, then define a prime token flow event structure $Ext(\mu, C, x, y, t) = ((PES', l'), x')$, $PES' = (E', Con', \prec')$, updating Con by e_{new} through $E' = E$, $l' = l$, $\prec' = \prec$, $Con' = Con \cup \{C' \cup \{e_{id}\} \mid C' \subseteq C\}$ and $\forall C' \in Con'_{pre} \setminus Con_{pre}$, $e_{id} \in C' : x'_{C'}|_{\prec} = x \wedge x'_{C'}|_{C' \times \{e_{id}\}} = x'$. We say the μ is updated by $e_{x,y}$.

In the reduced token flow unfolding, only one event having a certain set of pre-events is introduced (except for concurrent events in one run), although there are different possible distributions of the token flows on ingoing edges of the event. Only one example distribution of these possible token flows is stored.

For the reduced token flow unfolding analogous results hold as for token flow unfoldings. By construction $Ext(\mu, C, x, y, t)$ is a reduced token flow unfolding. Similar as for token flow unfoldings, a prefix relation $\sqsubseteq$ between reduced token flow unfoldings can be defined. Since through appending events, the token flow on old consistency sets is not changed, $\mu \sqsubseteq Ext(\mu, C, x, y, t)$ holds. Moreover, every finite reduced token flow unfolding μ can be constructed by a sequence of appending operations from $\mu_0 = (((\{e_{init}\}, \{\{e_{init}\}\}, \emptyset), id), \emptyset)$, where C, x, y and t are chosen according to μ .

Appending events to a token flow unfolding in different orders leads to reduced token flow unfoldings with isomorphic underlying prime event structures. Isomorphic prefixes (in different such reduced token flow unfoldings) may have different token flow distributions, representing processes with isomorphic underlying runs. In this sense, the order of appending events plays no role and we can define a maximal (w.r.t. $\sqsubseteq$) reduced token flow unfolding $Unf_{red}(N, m_0)$ as the limit of a sequence of finite token flow unfoldings $(\mu_n)_{n \in \mathbb{N}}$ with $\mu_{n+1} = Ext(\mu_n, C, x, y, t)$. $Unf_{red}(N, m_0)$ is unique up to isomorphism of the underlying prime event structure and up to the token flow stored for a consistency set, where different possible token flows produce isomorphic runs.

Each left-closed consistency set C of $Unf_{red}(N, m_0)$ represents a process π_C of (N, m_0) in the sense that the LPO $lpo_C = (C, \prec|_{C \times C}, l|_C)$ is the run underlying π_C and x_C is the canonical token flow function of π_C . Moreover, in $Unf_{red}(N, m_0)$ all runs underlying a process of (N, m_0) are represented by consistency sets (without loss of generality we can start the construction of $Unf_{red}(N, m_0)$ with an arbitrary process representing a specific run).

$Unf_{red}(N, m_0)$ avoids the generation of processes with isomorphic underlying runs. Namely, only in special auto-concurrency situations runs of (N, m_0) are represented more than once in $Unf_{red}(N, m_0)$. It can be seen similar as for token flow unfoldings that, if there are two sets $C \neq C' \in Con_{pre}$ representing processes with isomorphic underlying runs, then there exist events $e \in C, e' \in C', e \neq e'$ such that e and e' are weak identical and fulfill $\{e, e'\} \in Con$. That means, if (N, m_0) allows no auto-concurrency, there is a one-to-one correspondence between left-closed consistency sets of $Unf_{red}(N, m_0)$ and (isomorphism classes of) runs.

As a topic of future research, similar as for strong identical events and isomorphic processes, to avoid isomorphic runs, we can generalize the notion of weak identical events in the sense that two weak identical events have not necessarily common, but weak identical pre-events. This leads to the notion of *isomorphic weak identical* events analogously as for isomorphic strong identical events.

4 Algorithms

In this section we briefly describe two algorithms to construct unfolding models of a marked p/t-net with finite behavior. The algorithms essentially follow the Definitions 8 and 10. We implemented both methods.

The first algorithm computes a token flow unfolding equal to the maximal token flow unfolding, except that some isomorphic processes caused by auto-concurrency of transitions are omitted. Starting with the token flow unfolding consisting only of the initial event e_{init} and having the only consistency set $\{e_{init}\}$, events are appended to maximal left-closed consistency sets in a breadth-first way. In each iteration step, the algorithm picks the next consistency set C and, for each transition $t \in T$, stores all enabling functions for appending a t -occurrence. The enabling functions can be computed from the residual token flows of events $e \in C$ and the residual marking of C . Finding all possible choices of enabling functions is a combinatorial problem. For each enabling function, a new event is generated and the old token flow unfolding is either extended or updated by the new event in a similar way as described in Definition 8. In contrast to Definition 8, in each appending step only maximal left-closed consistency sets are constructed. Therefore, in some special cases of auto-concurrency the described algorithm does not construct all possible isomorphic strong identical events. That is because not all left-closures of subsets of strong identical events are considered as consistency sets (if there are two concurrent strong identical events, one is appended first and the second is only appended to consistency sets including the first appended event). Therefore, in general the calculated token flow unfolding contains less events than the maximal token flow unfolding. But calculating these events (which are isomorphic strong identical to already appended events) would only lead to isomorphic processes (i.e. the unfolding computed by the algorithm still represents all processes) and would worsen the runtime.

The second algorithm computes a reduced token flow unfolding equal to the maximal reduced token flow unfolding, except that only processes with minimal causality are represented. Starting with the token flow unfolding consisting only of the initial event e_{init} and having the only consistency set $\{e_{init}\}$, the algorithm essentially appends events to prefixes of left-closed consistency sets in a breadth-first way. In each

iteration step, the algorithm picks the next consistency set C and tries to append each transition t to prefixes of C . The aim is to find all minimal prefixes which allow to append a t -occurrence. We say that a transition occurrence can be appended to a prefix of a consistency set C , if there exists a token flow function fulfilling (IN), (OUT) and (INIT) of the resulting LPO. In [11] a polynomial algorithm to check this and to construct such a token flow function in the positive case was presented. For each computed token flow function, a new event is generated and the old token flow unfolding is either extended or updated by the new event in a similar way as described in Definition 8 (the computed token flow function defines the redistribution function and the enabling function). Since the algorithm appends transition occurrences only to minimal prefixes of C for which this is possible, the resulting reduced token flow unfolding contains all runs with minimal causality of the given p/t-net. In this sense it represents the complete partial order behavior. Observe that weak identical events are only constructed in cases of auto-concurrency, and that only left-closed consistency sets are constructed (each appending step leads a maximal left-closed consistency set). In contrast to the construction algorithm of the token flow unfolding, here all isomorphic runs appearing through auto-concurrency of events are computed, because all left-closures of subsets of weak identical events are considered as prefixes of consistency sets.

5 Experimental Results

In this section we experimentally test our implementation of the construction algorithms having the standard unfolding algorithm as a benchmark.

To construct the standard unfolding, we use an adapted version of the unfolding algorithm in [4]. When interpreting the results, one has to pay attention that this unfolding

				Standard unfolding				Token flow unfolding				Reduced token flow unfolding			
n	m	x	y	E	P	time	mem	E	P	time	mem	E	P	time	mem
1	3	2	3	19	12	82ms	1133kb	5	2	25ms	557kb	4	1	42ms	625kb
2	4	2	3	267	132	1406ms	2548kb	15	6	43ms	667kb	9	4	76ms	865kb
1	3	4	2	91	315	2721ms	2365kb	11	3	37ms	704kb	7	1	59ms	917kb
1	3	4	3	175	840	23622ms	4640kb	9	6	40ms	685kb	5	1	55ms	754kb
3	4	2	3	799	612	90665ms	5067kb	22	10	54ms	761kb	12	7	103ms	1160kb
3	4	4	3	-	-	-	-	43	56	271ms	2440kb	13	3	102ms	1159kb
3	4	5	3	-	-	-	-	45	104	672ms	6196kb	17	7	178ms	1149kb
n	m	k													
1	1	1		41	22	133ms	1011kb	13	6	47ms	834kb	13	6	103ms	1135kb
1	2	1		47	28	180ms	1270kb	13	6	49ms	841kb	13	6	114ms	1185kb
2	1	1		71	58	469ms	1325kb	17	9	65ms	917kb	15	7	145ms	1211kb
2	2	1		77	67	694ms	1438kb	17	9	65ms	917kb	15	7	145ms	1235kb
1	1	2		-	-	-	-	179	150	640ms	5200kb	71	68	8561ms	2580kb
2	2	2		-	-	-	-	239	413	3498ms	16991kb	95	147	54371ms	5414kb

Fig. 7. Experimental results: E shows the number of events and P the number of maximal processes in the constructed unfolding

algorithm is not completely runtime optimized, but the remaining improvement potential should be limited. We compare the runtime, memory consumption as well as the size and the number of maximal processes of the resulting event structures. The upper table in Figure 7 shows a test of the parameterized version of the example net of Figure 1 shown in Figure 8. The lower table in Figure 7 shows a test of the net in Figure 9 modeling for example a coffee automata.

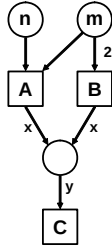


Fig. 8. Parameterized test net N_1

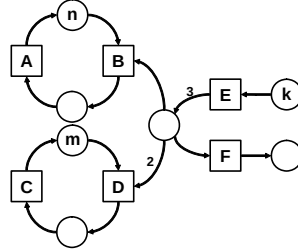


Fig. 9. Parameterized test net N_2

The experimental results indicate that our new unfolding approaches are superior to the standard approach. For the tested examples, the runtime, memory consumption and the sizes of the resulting structure of our new algorithms are a lot better. It is clear that the standard unfolding is least as big as the token flow unfolding and the reduced token flow unfolding, but usually considerably bigger, if the net contains arc weights or a non-safe initial marking. In these cases our new algorithms are significantly faster and use less memory. Comparing the two new approaches shows that in almost every tested case the calculated reduced token flow unfolding is actually smaller than the calculated token flow unfolding, but the redistribution of token flows in each step worsens the runtime.

6 Conclusion

In this paper we propose two new unfolding semantics for p/t-nets based on the concepts of prime event structures and token flows. The definitions of the two unfolding models are motivated by algorithmic aspects. We develop a construction algorithm for both unfolding models, if they are finite. We show that there are many cases in which our implemented algorithms are significantly more efficient than standard unfolding methods for p/t-nets.

We finally want to mention that the two presented unfolding models are a conservative extension of the unfolding model introduced in [5] for safe nets. That means, for safe nets, the standard unfolding, the token flow unfolding and the reduced token flow unfolding coincide.

Topic of further research is the application of isomorphic weak resp. strong identical events to avoid isomorphic runs resp. isomorphic processes at all, the adaption of the theory of complete finite prefixes to our approach and the adaption of model checking algorithms. Although there are complete finite prefixes which also avoid redundant events, we believe that our approach yields faster construction algorithms since such complete finite prefixes rely on complex adequate orders which cannot be implemented efficiently.

References

1. Best, E., Devillers, R.: Sequential and concurrent behaviour in petri net theory. *Theoretical Computer Science* 55(1), 87–136 (1987)
2. Couvreur, J.-M., Poitrenaud, D., Weil, P.: Unfoldings for general petri nets. University de Bordeaux I (Talence, France), University Pierre et Marie Curie (Paris, France) (2004), <http://www.labri.fr/perso/weil/publications/depliage.pdf>
3. Desel, J., G., Neumair, C.: Finite unfoldings of unbounded petri nets. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 157–176. Springer, Heidelberg (2004)
4. Desel, J., Juhás, G., Lorenz, R.: Viptool-homepage (2003), <http://www.informatik.ku-eichstaett.de/projekte/vip/>
5. Engelfriet, J.: Branching processes of petri nets. *Acta Informatica* 28(6), 575–591 (1991)
6. Esparza, J., Heljanko, K.: Implementing ltl model checking with net unfoldings. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 37–56. Springer, Heidelberg (2001)
7. Esparza, J., Römer, S., Vogler, W.: An improvement of mcmillan’s unfolding algorithm. *Formal Methods in System Design* 20(3), 285–310 (2002)
8. Goltz, U., Reisig, W.: The non-sequential behaviour of petri nets. *Information and Control* 57(2/3), 125–147 (1983)
9. Haar, S.: Branching processes of general s/t-systems and their properties. *Electr. Notes Theor. Comput. Sci.* 18 (1998)
10. Hoogers, P., Kleijn, H., Thiagarajan, P.: An event structure semantics for general petri nets. *Theoretical Computer Science* 153(1&2), 129–170 (1996)
11. Juhás, G., Lorenz, R., Desel, J.: Can i execute my scenario in your net? In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 289–308. Springer, Heidelberg (2005)
12. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged processes: a new condensed representation of petri net behaviour. *Acta Inf.* 43(5), 307–330 (2006)
13. Khomenko, V., Koutny, M.: Towards an efficient algorithm for unfolding petri nets. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 366–380. Springer, Heidelberg (2001)
14. Khomenko, V., Koutny, M.: Branching processes of high-level petri nets. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 458–472. Springer, Heidelberg (2003)
15. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of petri net unfoldings. *Acta Inf.* 40(2), 95–118 (2003)
16. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: von Bochmann, G., Probst, D.K. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)
17. Meseguer, J., Montanari, U., Sassone, V.: On the model of computation of place/transition petri nets. In: Valette, R. (ed.) ICATPN 1994. LNCS, vol. 815, pp. 16–38. Springer, Heidelberg (1994)
18. Meseguer, J., Montanari, U., Sassone, V.: On the semantics of place/transition petri nets. *Mathematical Structures in Computer Science* 7(4), 359–397 (1997)
19. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part i. *Theoretical Computer Science* 13, 85–108 (1981)
20. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)

Decomposition Theorems for Bounded Persistent Petri Nets

Eike Best¹ and Philippe Darondeau²

¹ Parallel Systems, Department of Computing Science
Carl von Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany
`eike.best@informatik.uni-oldenburg.de`

² IRISA, campus de Beaulieu, F-35042 Rennes Cedex
`darondeau@irisa.fr`

Abstract. We show that the cycles of a finite, bounded, reversible, and persistent Petri net can be decomposed in the following sense. There exists in the reachability graph a finite set of transition-disjoint cycles such that any other cycle through a given marking is permutation equivalent to a sequential composition of cycles from this set.

We show that Parikh images of cycles of a finite, bounded, and persistent Petri net form an additive monoid with a finite set of transition-disjoint generators (for any two distinct generators $\Psi(\gamma)$ and $\Psi(\gamma')$, $\Psi(\gamma)(t) = 0$ or $\Psi(\gamma')(t) = 0$ for every transition t).

Persistent nets are a very general class of conflict-free nets. Boundedness means, as usual, that the reachability graph is finite. Reversibility means that the reachability graph is strongly connected.

1 Introduction

Petri nets have traditionally been motivated by their ability to express concurrency. The subclass of finite Petri nets without concurrency may still exhibit conflict (choices, or nondeterministic alternatives) and much resembles the class of finite automata. Subclasses of Petri nets without conflicts have also been studied extensively. Perhaps the best known – and comparatively restricted – such class are the marked graphs [3].

There exists a well-known hierarchy of net classes encompassing marked graphs, of which free-choice nets [5] are a prominent one. Petri net structure theory has mainly been applied to this hierarchy. However, structure theory has been applied to a lesser degree to the persistent nets [8], a class of nets that is significantly larger than marked graphs since it contains all conflict-free nets. E.g., a manufacturing process in which three operations A,B,C are performed according to the cyclic workflow ABACABAC... may be described by a persistent net with three transitions but not by any marked graph with three transitions. Persistent nets are incomparable to free-choice nets – neither class is a subset of the other.

Some early results about persistent nets are Keller's theorem [7], which will be recalled in a later part of this paper, the famous semilinearity result of Landweber and Robertson [8], which states that the set of reachable markings of a persistent

net is semilinear, and Grabowski's proof of the decidability of persistence of vector addition systems [6]. Since then, however, some new open questions have arisen in this context.

It is the purpose of this paper to go some way towards solving such questions and, more generally, to help making structure theory more amenable to persistent nets. In particular, we show that for bounded and reversible persistent nets, a decomposition of the cycles of the reachability graph (and thus, of the realisable T-invariants) into smaller, disjoint cycles can be found. We propose a similar decomposition of the Parikh images of cycles for bounded persistent nets. The motivation of these developments is to try reducing the gap between the linear algebraic properties of nets (e.g. concerning T-invariants) and their more combinatorial properties (e.g. which T-invariants are realised by cycles).

2 Definitions

A Petri net (S, T, F, M_0) consists of two finite and disjoint sets S (places) and T (transitions), a function $F: ((S \times T) \cup (T \times S)) \rightarrow \mathbb{N}$ (flow) and a marking M_0 (the initial marking). A marking is a mapping $M: S \rightarrow \mathbb{N}$.

The incidence matrix C is an $S \times T$ -matrix of integers where the entry corresponding to a place s and a transition t is, by definition, equal to the number $F(t, s) - F(s, t)$. A T-invariant J is a vector of integers with index set T satisfying $C \cdot J = \underline{0}$ where $\cdot$ is the inner (scalar) product, and $\underline{0}$ is the vector of zeros with index set S . J is called semipositive if $J(t) \geq 0$, for all $t \in T$, and J is not the null vector. Two semipositive T-invariants J and J' are called transition-disjoint if $\forall t \in T: J(t) = 0 \vee J'(t) = 0$. For a sequence $\sigma \in T^*$ of transitions, the Parikh vector $\Psi(\sigma)$ is a vector of natural numbers with index set T , where $\Psi(\sigma)(t)$ is equal to the number of occurrences of t in σ .

A transition t is enabled (or activated, or firable) in a marking M (denoted by $M[t]$) if, for all places s , $M(s) \geq F(s, t)$. If t is enabled in M , then t can occur (or fire) in M , leading to the marking M' defined by $M'(s) = M(s) + F(t, s) - F(s, t)$ (notation: $M[t]M'$). We apply definitions of enabledness and of the reachability relation to transition (or firing) sequences $\sigma \in T^*$, defined inductively: $M[\varepsilon]$ and $M[\varepsilon]M$ are always true; and $M[\sigma t]$ (or $M[\sigma t]M'$) iff there is some M'' with $M[\sigma]M''$ and $M''[t]$ (or $M''[t]M'$, respectively).

A marking M is reachable (from M_0) if there exists a transition sequence σ such that $M_0[\sigma]M$. The reachability graph of N , with initial marking M_0 , is the graph whose vertices are the markings reachable from M_0 and where an edge labelled with t leads from M to M' iff $M[t]M'$. Figure 1 shows an example where on the right-hand side, M_0 denotes the marking shown in the Petri net on the left-hand side. The marking equation states that if $M[\sigma]M'$, then $M' = M + C \cdot \Psi(\sigma)$. Thus, if $M[\sigma]M$ then $\Psi(\sigma)$ is a T-invariant.

A Petri net is k -bounded if in any reachable marking M , $M(s) \leq k$ holds for every place s , and bounded if there is some k such that it is k -bounded. A finite Petri net (and we consider only such nets in the sequel) is bounded if and only if the set of its reachable markings is finite.

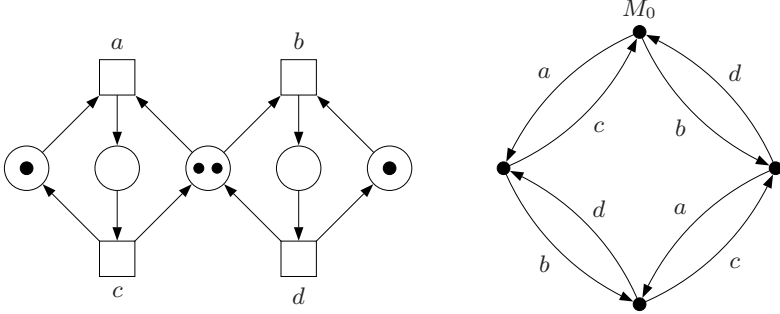


Fig. 1. A persistent Petri net (l.h.s.) and its reachability graph (r.h.s.)

A net $N = (S, T, F, M_0)$ is called output-nonbranching if $\sum_{t \in T} F(s, t) \leq 1$ for all places s , and a marked graph if $\sum_{t \in T} F(s, t) \leq 1$ as well as $\sum_{t \in T} F(t, s) \leq 1$, for all places s .

3 Persistent Nets, and Related Notions

A net N , with some initial marking, will be called persistent, if whenever $M[t_1\rangle$ and $M[t_2\rangle$ for a reachable marking M and transitions $t_1 \neq t_2$, then $M[t_1 t_2\rangle$. Note that output-nonbranching nets, and *a fortiori*, marked graphs, are automatically persistent, for any initial marking.

Two sequences $M[\sigma\rangle$ and $M[\sigma'\rangle$, firable from M , are said to arise from each other by a transposition if they are the same, except for the order of an adjacent pair of transitions, thus:

$$\sigma = t_1 \dots t_k t t' \dots t_n \quad \text{and} \quad \sigma' = t_1 \dots t_k t' t \dots t_n.$$

Two sequences $M[\sigma\rangle$ and $M[\sigma'\rangle$ are said to be permutations of each other (from M , written $\sigma \equiv_M \sigma'$) if they are both firable at M and arise out of each other through a (possibly empty) sequence of transpositions.

Note that the permuted transitions t and t' do not need to be concurrently enabled by the marking preceding them, i.e., the marking reached after $M[t_1 \dots t_k\rangle$. For example, in a net with one single-token place s and two transitions t and t' that are both input and output transitions of s (with arc weight 1), firing sequences tt' and $t't$ are permutations of each other.

By $\tau \dot{-} \sigma$, we mean the residue of τ left after cancelling successively in this sequence the leftmost occurrences of all symbols from σ , read from left to right. Formally, $\tau \dot{-} \sigma$ can be defined by induction on the length of σ :

$$\begin{aligned} \tau \dot{-} \varepsilon &= \tau \\ \tau \dot{-} t &= \begin{cases} \tau, & \text{if there is no transition } t \text{ in } \tau \\ \text{the sequence obtained by erasing the leftmost } t \text{ in } \tau, & \text{otherwise} \end{cases} \\ \tau \dot{-} (t\sigma) &= (\tau \dot{-} t) \dot{-} \sigma. \end{aligned}$$

From this definition, it should be clear that $\tau \dot{-} \sigma$ and $\sigma \dot{-} \tau$ contain no common transitions. It may be noted that the residual operation $\dot{-}$ satisfies the following two properties, reminiscent of the axiomatic properties of the residual operation $\uparrow$ in the concurrent transition systems studied in [9]:

$$\begin{aligned}\tau \dot{-} (\sigma \sigma') &= (\tau \dot{-} \sigma) \dot{-} \sigma' \\ (\tau \tau') \dot{-} \sigma &= (\tau \dot{-} \sigma) (\tau' \dot{-} (\sigma \dot{-} \tau))\end{aligned}$$

One may therefore expect that the set of firing sequences of a persistent net forms a concurrent transition system with a residual operation of the form $M[\tau]M_1 \uparrow M[\sigma]M_2 = M_2[\tau \dot{-} \sigma]M_3$. This is essentially what the following theorem shows.

Theorem 1. KELLER [7]

Let N be a persistent net and let τ and σ be two firing sequences starting from some reachable marking M . Then

$$\tau(\sigma \dot{-} \tau) \text{ and } \sigma(\tau \dot{-} \sigma)$$

are also firing sequences from M , and what is more, they are permutations of each other, i.e., $\tau(\sigma \dot{-} \tau) \equiv_M \sigma(\tau \dot{-} \sigma)$. In particular, the marking reached after $\tau(\sigma \dot{-} \tau)$ equals the marking reached after $\sigma(\tau \dot{-} \sigma)$.

As the above (Keller's) theorem is the basis for our subsequent results, we include a proof of this theorem in the Appendix. In the end of the present section, we establish a series of consequences of Theorem 1.

Corollary 1. DIRECTEDNESS OF PERSISTENT NETS

Let N be a persistent net and let M_1 and M_2 be two reachable markings. Then there is some marking M_3 which is reachable from M_1 and from M_2 .

Proof: By the definition of reachability, there are two sequences τ and σ leading from the initial marking M_0 to M_1 and to M_2 , respectively. The claim follows from Theorem 1, applied to $M = M_0$. $\square$

If two sequences are permutations of each other (from some marking), then they have the same Parikh vector. In persistent nets, the converse is also true:

Lemma 1. PARIKH-EQUIVALENCE AND $\equiv$

Let N be a persistent net, M a reachable marking, and $M[\tau]$ and $M[\sigma]$ two transition sequences which are firable at M . If $\Psi(\tau) = \Psi(\sigma)$, then $\tau \equiv_M \sigma$.

Proof: By the definitions of Parikh vector and $\dot{-}$, $\Psi(\tau) = \Psi(\sigma)$ if and only if both $\tau \dot{-} \sigma = \varepsilon$ and $\sigma \dot{-} \tau = \varepsilon$. The lemma follows by Theorem 1. $\square$

Lemma 2. A PERMUTATION LEMMA

Let N be persistent and let $M[\gamma]$ and $M[\kappa\gamma]$ be two firing sequences from M , with transition sequences γ and κ .

Then there is a firing sequence $M[\gamma\kappa']$ such that $\Psi(\kappa) = \Psi(\kappa')$ and $\kappa\gamma \equiv_M \gamma\kappa'$.

That is, κ can as a whole be permuted with γ , albeit, possibly, up to transition re-orderings within κ .

Proof: By Keller's theorem, $M[\gamma]$ and $M[\kappa\gamma]$ imply that $M[\gamma(\kappa\gamma \bullet \gamma)]$. Put $\kappa' = \kappa\gamma \bullet \gamma$. Clearly, $\Psi(\kappa\gamma \bullet \gamma) = \Psi(\kappa)$, hence $\Psi(\gamma\kappa') = \Psi(\kappa\gamma)$. By Lemma 1, $\kappa\gamma \equiv_M \gamma\kappa'$ follows since both sequences are firable from M . $\square$

Lemma 3. DISJOINT SEQUENCES CONTRADICT BOUNDEDNESS

Let N be bounded and persistent and let τ and σ be two firing sequences, both firable at M and leading to the same marking $\widetilde{M}$. Further, suppose that $M \neq \widetilde{M}$. Then there is at least one transition which occurs both in τ and in σ .

Proof: We will prove the lemma by contradiction. First, note that by $M \neq \widetilde{M}$ and because τ and σ lead to $\widetilde{M}$, $\tau \neq \varepsilon \neq \sigma$. Assume that τ and σ are transition-disjoint. Then $\tau \bullet \sigma = \tau$ and $\sigma \bullet \tau = \sigma$. By (the proof of) Keller's theorem, we get a special case of the diamond in Figure 7, namely Diamond 1 in Figure 2. By assumption, the markings reached after τ and σ are the same, so that the West and East corners of Diamond 1 are the same marking $\widetilde{M}$. Because σ is firable from the West corner, it is also firable from the East corner, and thus we get Diamond 2, again from transition-disjointness. The West and East corners of Diamond 2 are again the same, because both are the marking obtained by firing σ from $\widetilde{M}$. In this way, we get an infinite sequence $M[\sigma]\widetilde{M}[\sigma]\widetilde{M}[\sigma]\widetilde{M}[\sigma]\dots$ along the Eastern ridge of the diamonds. Moreover, since $M \neq \widetilde{M}$, there is some place s with $M(s) \neq \widetilde{M}(s)$. Since the effect of σ is monotonic, it must be the case that both $\widetilde{M}(s) \neq \widehat{M}(s)$ and $M(s) \neq \widehat{M}(s)$. Thus we also have that $\widehat{M}$ is not in $\{M, \widetilde{M}\}$, and that $\widehat{M}$ is not in $\{M, \widetilde{M}, \widehat{M}\}$, and so on. However, this contradicts boundedness. Hence, our assumption was wrong, and instead, the claim of the lemma is true. $\square$

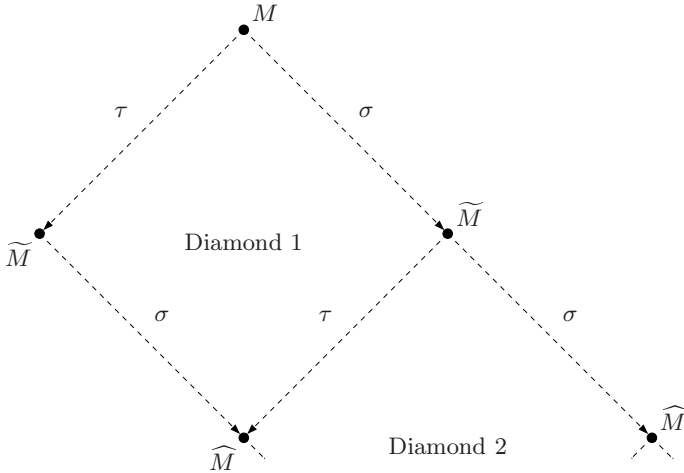


Fig. 2. Completing diamonds, starting from M

The principle used in the proof of Lemma 3 is typical of bounded persistent nets and it served also for proving Prop. 8 in [4].

In the sequel, we focus on cycles in reachability graphs of bounded persistent nets. Section 4 provides a general analysis of cycles. Section 5 proposes a sequential decomposition of cycles in bounded persistent and reversible nets. Section 6 proposes an algebraic decomposition of cycles in bounded persistent nets.

4 Analysis of Cycles in the Reachability Graph

A transition sequence τ is called cyclic if $C \cdot \Psi(\tau) = \underline{0}$, i.e. if $\Psi(\tau)$ is a T-invariant. From the marking equation, τ is cyclic if and only if $M[\tau]M$, for all markings M which activate τ .

Definition 1. DECOMPOSABILITY AND SIMPLICITY

A cyclic transition sequence τ is called decomposable if $\tau = \tau_1\tau_2$ such that τ_1 and τ_2 are cyclic and $\tau_1 \neq \varepsilon \neq \tau_2$.

A firing sequence $M[\tau]M'$ is called a cycle if τ is cyclic, i.e. if $M = M'$.

A cycle $M[\tau]M$ is called simple if there is no permutation $\tau' \equiv_M \tau$ such that τ' is decomposable. $\square$

In other words, a non-simple sequence can be permuted such that the permuted sequence has a smaller cyclic subsequence leading from some marking back to the same marking. In Figure 1, for example, we have that:

$abcd$	is not decomposable
$acbd$	is decomposable, namely by $\tau_1 = ac$ and $\tau_2 = bd$
$M_0[ac]M_0$	is simple
$M_0[abcd]M_0$	is not simple, because of the permutation $M_0[acbd]M_0$.

Figure 1 demonstrates that two simple cycles in the reachability graph can be different, even if they start at the same marking. For example, there are two transition-disjoint simple cycles through M_0 , viz. $M_0[ac]M_0$ and $M_0[bd]M_0$. We now show that in general, if two such simple cycles start with the same transition, then they have the same set and the same number of transitions, i.e., their Parikh vectors are identical.

Lemma 4. NONDISJOINT SIMPLE CYCLES ARE UNIQUE, UP TO PERMUTATION

Let N , with some initial marking, be bounded and persistent. Let $M[a\tau]M$ and $M[a\sigma]M$, with some transition a and transition sequences τ and σ (see Figure 3). Suppose that both $M[a\tau]M$ and $M[a\sigma]M$ are simple. Then $a\tau \equiv_M a\sigma$.

Proof: First, we prove that $\tau \bullet \sigma = \varepsilon$ implies $\sigma \bullet \tau = \varepsilon$. To see this, suppose that we have $\tau \bullet \sigma = \varepsilon$ and $\sigma \bullet \tau \neq \varepsilon$; we derive a contradiction.

By Keller's theorem, $a\tau(\sigma \bullet \tau)$ is fireable from M and permutes $a\sigma(\tau \bullet \sigma)$, from M . Because $a\sigma = a\sigma(\tau \bullet \sigma)$ reproduces M , so does $a\tau(\sigma \bullet \tau)$. But because $a\tau$ reproduces M as well, we have

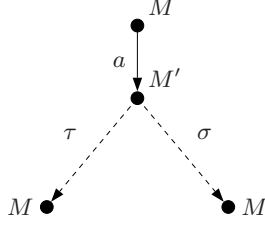


Fig. 3. Two simple cycles with the same initial transition a

$$M \underbrace{[a\tau]}_{\neq \varepsilon} \rangle M \underbrace{[\sigma^{\bullet}\tau]}_{\neq \varepsilon} \rangle M.$$

But this contradicts the simplicity of $M[a\sigma]M$, since $a\sigma \equiv_M a\tau(\sigma^{\bullet}\tau)$.

By symmetry, the proof of the lemma can be split into two separate cases:

$$\tau^{\bullet}\sigma = \varepsilon = \sigma^{\bullet}\tau \quad \text{or} \quad \tau^{\bullet}\sigma \neq \varepsilon \neq \sigma^{\bullet}\tau.$$

Case 1: $\tau^{\bullet}\sigma = \varepsilon = \sigma^{\bullet}\tau$.

Then we have $\Psi(\tau) = \Psi(\sigma)$, whence also $\Psi(a\tau) = \Psi(a\sigma)$, and the desired result $a\tau \equiv_M a\sigma$ follows directly from Lemma 1.

Case 2: $\tau^{\bullet}\sigma \neq \varepsilon \neq \sigma^{\bullet}\tau$.

We will derive a contradiction, showing that actually only Case 1 remains, and thus proving the lemma.

Transition a cannot occur in both $\tau^{\bullet}\sigma$ and $\sigma^{\bullet}\tau$, since the two sequences have no common transitions. Without loss of generality, we may assume that a does not occur in $\tau^{\bullet}\sigma$. Next, we prove the following in turn:

1. The sequences $\sigma^{\bullet}\tau$ and $\tau^{\bullet}\sigma$ are both firable from M , and when fired from $\widetilde{M}$, they lead to the same marking, say to $\widetilde{M}$.
2. $\widetilde{M} \neq M$.

When this is proved, we may use boundedness and apply Lemma 3, yielding the result that $\sigma^{\bullet}\tau$ and $\tau^{\bullet}\sigma$ must have some transition in common. This is a contradiction to their definition.

Proof of 1.: Both $a\tau$ and $a\sigma$ are firable from M , leading to M . By Keller's theorem, $a\tau(a\sigma^{\bullet}a\tau)$ and $a\sigma(a\tau^{\bullet}a\sigma)$ are firable from M and moreover, they lead to the same marking, $\widetilde{M}$. Thus, $a\tau^{\bullet}a\sigma$ and $a\sigma^{\bullet}a\tau$ can both be fired from M and they lead to $\widetilde{M}$. Clearly $a\sigma^{\bullet}a\tau = \sigma^{\bullet}\tau$ and $a\tau^{\bullet}a\sigma = \tau^{\bullet}\sigma$, therefore $M[\sigma^{\bullet}\tau]\widetilde{M}$ and $M[\tau^{\bullet}\sigma]\widetilde{M}$.

Proof of 2.: By contradiction. Assume that $\widetilde{M} = M$.

First of all, we have the following reproducing firing sequence:

$$M[\tau^{\bullet}\sigma]M, \tag{1}$$

simply by Part 1. above, and by $\widetilde{M} = M$. Note that $\tau \dot{\sigma}$ is just the sequence of transitions (in the correct order) from τ that are *not* matchable with transitions in σ , in the sense of (our proof of) Keller's theorem.

Next, we will show that the following is also a firing sequence:

$$M[a(\tau \dot{\sigma}(\tau \dot{\sigma})) \rangle M. \quad (2)$$

Note that $\tau \dot{\sigma}(\tau \dot{\sigma})$ is exactly (again in the correct order) the remaining sequence of transitions from τ , i.e. the ones that *are* matchable with transitions from σ .

We now prove (2). Let $M[a \rangle M'$ (Figure 3). By $M[a \rangle M'$ and $M[\tau \dot{\sigma} \rangle M$ (Equation (1)), Keller's theorem yields

$$M[a \rangle M'[(\tau \dot{\sigma}) \dot{a}]. \quad (3)$$

Because a is not contained in $\tau \dot{\sigma}$, we have $((\tau \dot{\sigma}) \dot{a}) = (\tau \dot{\sigma})$. Hence, $\tau \dot{\sigma}$ is firable from M' . Moreover, because $\tau \dot{\sigma}$ reproduces M , $\Psi(\tau \dot{\sigma})$ is a T-invariant, and the second part of Equation (3) yields

$$M'[\tau \dot{\sigma} \rangle M'. \quad (4)$$

By Equation (4) and $M'[\tau \rangle M$ (cf. Figure 3), and by Keller's theorem,

$$M'[\tau \dot{\sigma} \rangle M'[\tau \dot{\sigma}(\tau \dot{\sigma}) \rangle M. \quad (5)$$

The final marking is indeed M , since by Keller's theorem, it must coincide with the marking reached by the firing sequence $M'[\tau \rangle M[(\tau \dot{\sigma}) \dot{a}]$ and $(\tau \dot{\sigma}) \dot{a} = \varepsilon$. Combining $M[a \rangle M'$ (cf. Figure 3) and Equations (1) and (5), we have the firing sequence

$$M[(\tau \dot{\sigma}) \rangle M[a(\tau \dot{\sigma}(\tau \dot{\sigma})) \rangle M. \quad (6)$$

The second part of (6) proves (2). Moreover, the sequence (6) is decomposable, because both parts are nonempty: $\tau \dot{\sigma} \neq \varepsilon$ by assumption, and $a(\tau \dot{\sigma}(\tau \dot{\sigma})) \neq \varepsilon$ because a is contained in it. But because the sequence $M[(\tau \dot{\sigma}) \dot{a}(\tau \dot{\sigma}(\tau \dot{\sigma})) \rangle M$ is a permutation of $M[a(\tau \dot{\sigma})(\tau \dot{\sigma}(\tau \dot{\sigma})) \rangle M$ and hence also of $M[a \rangle M$ by construction and by Lemma 1, we get a contradiction to the simplicity of $M[a \rangle M$.

Hence the assumption was wrong, and $\widetilde{M} \neq M$ must hold true. This ends the proof of 2., and thus also the proof of the lemma. $\square$

5 Sequential Decomposition of Cycles

For a large class of persistent nets, Lemma 4 leads to a decomposition theorem. Call a net with initial marking reversible if its reachability graph is strongly connected, i.e., if its initial marking is reachable from every reachable marking. First, we show a strengthened version of Lemma 4.

Lemma 5. SIMPLE CYCLES WITH COMMON TRANSITIONS ARE PARIKH-EQUIVALENT
Let N , with some initial marking, be bounded, reversible, and persistent. Moreover, let M, M' be two reachable markings and let $M[a \rangle M$ and $M'[a \sigma \rangle M'$ be two simple cycles (not necessarily $M = M'$). Then $\Psi(a \tau) = \Psi(a \sigma)$.

Proof: We have

$$\begin{aligned} 1 & M[a\tau]M[\xi]M'[a\sigma]M'[\chi]M \\ 2 & M[\xi]M'[\rho]M'[a\sigma]M'[\chi]M \quad \text{with } \Psi(\rho) = \Psi(a\tau) \end{aligned}$$

Line 1 uses the assumptions, and sequences ξ and χ exist because of strong connectedness. Line 2 follows by Lemma 2 from M , with $\kappa = a\tau$ and $\gamma = \xi$. We use here the fact that $a\tau$ reproduces M , as the preconditions $M[a\tau\xi]$ and $M[\xi]$ of Lemma 2 follow from the first M and the second M , respectively.

Now suppose that $M'[\rho]M'$ is a simple cycle. Using the fact that M' enables a we can move the first a in ρ to the front, getting a simple cycle of the form $M'[a\rho']M'$ with $a\rho' \equiv_{M'} \rho$. Then the claim of the lemma follows immediately from Lemma 4, the fact that $M'[a\sigma]M'$ is also a simple cycle, and the fact that $\Psi(a\rho') = \Psi(\rho) = \Psi(a\tau)$.

Otherwise, $M'[\rho]M'$ is not a simple cycle and can be split in this way:

$$M'[\rho_1]M'[\rho_2]M', \quad \text{with } \rho \equiv_{M'} \rho_1\rho_2 \text{ and } \rho_1 \neq \varepsilon \neq \rho_2.$$

We derive a contradiction. Since a is in ρ , it is also in $\rho_1\rho_2$. W.l.o.g., we may assume one occurrence of a to be in ρ_2 (otherwise, we may just exchange the roles of ρ_1 and ρ_2 because $\rho_1\rho_2 \equiv_{M'} \rho_2\rho_1$ by Lemma 2), and we may also assume that $M'[\rho_2]M'$ is simple (otherwise we just keep splitting and move an appropriate one of the simple cycles so obtained to the end). Then we may continue the above sequence as follows:

$$\begin{aligned} 3 & M[\xi]M'[\rho_1]M'[\rho_2]M'[a\sigma]M'[\chi]M \quad \text{with } \rho \equiv_{M'} \rho_1\rho_2 \\ 4 & M[\xi]M'[\rho_1]M'[a\rho'_2]M'[a\sigma]M'[\chi]M \quad \text{with } \Psi(a\rho'_2) = \Psi(\rho_2) = \Psi(a\sigma) \\ 5 & M[\xi]M'[\rho_1]M'[a\sigma]M'[\tilde{\rho}_2]M'[\chi]M \quad \text{with } \Psi(\tilde{\rho}_2) = \Psi(a\rho'_2) \\ 6 & M[\xi]M'[\rho_1]M'[a\sigma]M'[\chi]M[\tilde{\rho}_2']M \quad \text{with } \Psi(\tilde{\rho}_2') = \Psi(\tilde{\rho}_2) = \Psi(a\rho'_2) \end{aligned}$$

Line 3 simply follows from line 2 by the definitions of ρ_1 and ρ_2 . Line 4 comes from line 3 by Lemma 2 from M' , with κ being the initial a -free part of ρ_2 and $\gamma = a$. The fact that $\Psi(a\rho'_2) = \Psi(a\sigma)$ comes from Lemma 4, since both $M'[a\rho'_2]M'$ and $M'[a\sigma]M'$ are actually simple cycles. Line 5 is another application of Lemma 2 from M' , with $\kappa = a\rho'_2$ and $\gamma = a\sigma$. Finally, line 6 stems from line 5 by Lemma 2 with $\kappa = \tilde{\rho}_2$ and $\gamma = \chi$.

Line 6 shows that M can be reproduced by $\tilde{\rho}_2'$. By assumption, M can also be reproduced by $a\tau$. By Keller's theorem, both $\tilde{\rho}_2'(a\tau \dot{-} \tilde{\rho}_2')$ and $a\tau(\tilde{\rho}_2' \dot{-} a\tau)$ are firable from M , and moreover, $\tilde{\rho}_2'(a\tau \dot{-} \tilde{\rho}_2') \equiv_M a\tau$, since $\tilde{\rho}_2' \dot{-} a\tau = \varepsilon$ by the definition of $\tilde{\rho}_2'$ (tracing back through the above, one sees that $\tilde{\rho}_2'$ permutes a proper subsequence of ρ and hence of $a\tau$). This is clearly a decomposition, since $\tilde{\rho}_2'$ is nonempty and $a\tau \dot{-} \tilde{\rho}_2'$ is nonempty as well, for it has the same Parikh vector as ρ_1 . This contradicts the simplicity of $M[a\tau]M$. $\square$

Next, we strengthen Lemma 5 to the case where the simple cycles under consideration do not start with the same transition.

Lemma 6. ADVANCING IN A CYCLE PRESERVES SIMPLENESS

Let N be a bounded, reversible, persistent net. Let M be a reachable marking and $M[t_1 \dots t_n]M$ a simple cycle with $n \geq 1$. Let $M[t_1]\widehat{M}$.

Then the cycle $\widehat{M}[t_2 \dots t_n]M[t_1]\widehat{M}$ is also simple.

Proof: We prove the lemma by induction on n .

Base: $n = 1$. Then $M = \widehat{M}$, and the claim is trivially true.

Step: $n > 1$. Assume that the claim holds for all $m < n$; we prove it by contradiction for n .

Assume that $\widehat{M}[t_2 \dots t_n]M[t_1]\widehat{M}$ is not simple. Then there exist $\tau_1 \neq \varepsilon \neq \tau_2$ with $t_2 \dots t_n t_1 \equiv_{\widehat{M}} \tau_1 \tau_2$ and $\widehat{M}[\tau_1]\widehat{M}[\tau_2]\widehat{M}$. Because $t_2 \dots t_n t_1 \equiv_{\widehat{M}} \tau_1 \tau_2$, transition t_1 occurs somewhere in $\tau_1 \tau_2$. Without loss of generality, we may assume that it occurs in τ_1 (otherwise the roles of τ_1 and τ_2 can be exchanged). Moreover, we can assume that $\widehat{M}[\tau_1]\widehat{M}$ is simple, otherwise we continue splitting and move one of the τ 's so obtained to the front if it contains t_1 (this is possible since all intermediate markings equal $\widehat{M}$).

Thus, we get $\tau_1 = \alpha t_1 \beta$ with

$$\underbrace{\widehat{M}[\alpha]M'[t_1\beta]\widehat{M}}_{\text{simple cycle } \widehat{M}[\tau_1]\widehat{M}}[\alpha]M'.$$

The cycle $\widehat{M}[\tau_1]\widehat{M}$ is shorter than the original cycle with n transitions, because $\tau_2 \neq \varepsilon$. Applying the induction hypothesis $|\alpha|$ times to it, we get the result that

$$M' [t_1 \beta \alpha] M'$$

is also a simple cycle. Combining this with the premise that $M[t_1 \dots t_n]M$ is a simple cycle, Lemma 5 yields $\Psi(t_1 \dots t_n) = \Psi(t_1 \beta \alpha)$. However, since $\Psi(t_1 \beta \alpha) = \Psi(\tau_1)$, we get $\Psi(t_1 \dots t_n) = \Psi(\tau_1)$, and hence a contradiction to the fact that $t_2 \dots t_n t_1 \equiv_{\widehat{M}} \tau_1 \tau_2$ and $\tau_2 \neq \varepsilon$. $\square$

Lemma 7. SIMPLE CYCLES ARE EITHER DISJOINT OR PARIKH-EQUIVALENT

Let N be bounded, reversible, and persistent, let M and M' be reachable markings, and let $M[\rho]M$ and $M'[\rho']M'$ be two simple cycles.

Then either ρ and ρ' have no transition in common, or $\Psi(\rho) = \Psi(\rho')$.

Proof: If ρ and ρ' have no transition in common, there is nothing more to prove. Otherwise, suppose that $\rho = \rho_1 t \rho_2$ and $\rho' = \rho'_1 t \rho'_2$. Then we have two firing sequences

$$M[\rho_1]\widehat{M}[t\rho_2]M[\rho_1]\widehat{M} \quad \text{and} \quad M'[\rho'_1]\widehat{M}'[t\rho'_2]M'[\rho'_1]\widehat{M}'.$$

By Lemma 6 (possibly applied several times), $\widehat{M}[t\rho_2\rho_1]\widehat{M}$ and $\widehat{M}'[t\rho'_2\rho'_1]\widehat{M}'$ are simple cycles. By Lemma 5, $\Psi(t\rho_2\rho_1) = \Psi(t\rho'_2\rho'_1)$. The claim follows immediately. $\square$

Lemma 7 allows us to formulate and prove the main result of this section.

Theorem 2. FIRST DECOMPOSITION THEOREM

Let N , with some initial marking, be bounded, reversible, and persistent.

There is a finite set $\mathcal{B}$ of semipositive T -invariants such that any two of them are transition-disjoint and every cycle $M[\rho]M$ in the reachability graph decomposes up to permutations to some sequence of cycles $M[\rho_1]M[\rho_2]M \dots [\rho_n]M$ with all Parikh vectors $\Psi(\rho_i)$ in $\mathcal{B}$.

Proof: We construct $\mathcal{B}$ as follows: for each transition t occurring in the reachability graph, we choose a simple cycle through t (such a cycle always exists by strong connectedness) and we let the Parikh vector of this simple cycle be an element of $\mathcal{B}$. By Lemma 7, the vector is independent of the choice of where t occurs in the reachability graph, and also of the cycle that was chosen.

The set $\mathcal{B}$ so constructed is finite since there are only finitely many transitions. By Lemma 7, the Parikh vectors obtained for two different transitions are either equal or disjoint. This guarantees the transition-disjointness of $\mathcal{B}$.

Since every cycle in the reachability graph can be (permuted and then) decomposed into simple cycles, the claim follows. $\square$

We also establish a kind of converse of this theorem. First, we show that every simple cycle can be realised (albeit perhaps in a different order) at every reachable marking.

Lemma 8. EVERY SIMPLE CYCLE MAY BE FIRED EVERYWHERE

Let N , with some initial marking, be bounded, reversible, and persistent. Let M and M' be reachable markings and let $M[\alpha]M$ be a simple cycle.

Then there is a cycle $M'[\alpha']M'$ such that $\Psi(\alpha) = \Psi(\alpha')$.

Proof: The claim is trivially true if $\alpha = \varepsilon$. So, let $M[\alpha]M$ be a simple cycle with $\alpha \neq \varepsilon$. By reversibility, M' is reachable from M . Let $M[\gamma_0]M'$. By Keller's theorem, we have

$$M[\gamma_0]M'[\alpha \dot{-} \gamma_0]\widehat{M}. \quad (7)$$

If $\alpha \dot{-} \gamma_0 \neq \varepsilon$, then M' enables some transition t contained in α . Let $M'[\alpha']M'$ be any simple cycle such that α' contains t . (Such a cycle exists by reversibility.) Then $\Psi(\alpha) = \Psi(\alpha')$ by Theorem 2. Thus in this case, the assertion of the lemma is proved.

Suppose, on the other hand, that $\alpha \dot{-} \gamma_0 = \varepsilon$. By Keller's theorem, we also have

$$M[\alpha]M[\gamma_0 \dot{-} \alpha]\widehat{M} = M'.$$

Indeed $\widehat{M} = M'$ by $\alpha \dot{-} \gamma_0 = \varepsilon$ and by (7). Define $\gamma_1 = \gamma_0 \dot{-} \alpha$. By this definition and by $\alpha \dot{-} \gamma_0 = \varepsilon$ and $\alpha \neq \varepsilon$, γ_1 is strictly shorter than γ_0 and also leads from M to M' . Now we can repeat the argument with γ_1 instead of γ_0 .

Eventually, we will have $\alpha \dot{-} \gamma_i \neq \varepsilon$ for one of the γ 's so defined, proving the result. $\square$

We get the following immediate consequence from Lemma 8 and the construction of $\mathcal{B}$.

Corollary 2. A CONVERSE OF THEOREM 2

Let N , with some initial marking, be bounded, reversible, and persistent. Let M be a reachable marking. Let $b_1, \dots, b_n$ be (not necessarily mutually distinct) T -invariants from $\mathcal{B}$, constructed in the proof of Theorem 2.

Then there is a cycle $M[\rho]M$ such that $\Psi(\rho) = \sum_{i=1}^n b_i$. $\square$

6 Algebraic Decomposition of Cycles

Lemmas 5, 7 and 8, Theorem 2 and Corollary 2 do not apply to all bounded persistent nets, but only to reversible ones.

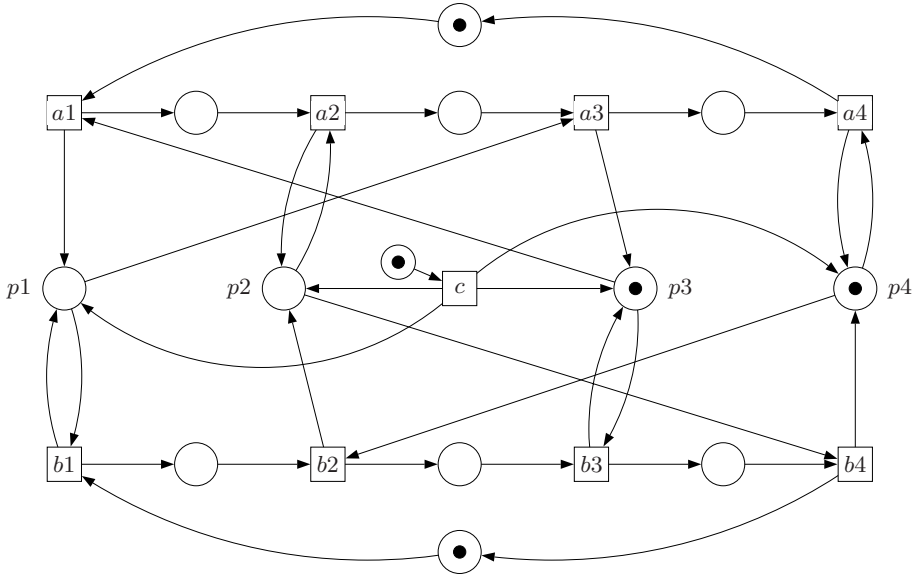


Fig. 4. A bounded persistent net which is not reversible

To assess the role of reversibility, one may consider the net shown in Figure 4. It is formed by connecting the two cyclic subnets spanned by $a_1, \dots, a_4$ and $b_1, \dots, b_4$ by four places $p_1, \dots, p_4$. Initially (i.e. before firing transition c), these places serve to synchronise the two cycles in a lock-step manner.

As long as transition c has not been fired, which may be done at any time but only once, the behaviour of this net is cyclic, and the unique cycle starting from the initial marking M_0 is $M_0[a_1 \cdot b_1 \cdot b_2 \cdot a_2 \cdot a_3 \cdot b_3 \cdot b_4 \cdot a_4]M_0$. All markings met in this cycle project along p_1, p_2, p_3, p_4 on vectors in the set $\{0011, 1001, 1100, 0110\}$.

Suppose now that the transitions from the two cyclic net components to be fired next are a_i and b_j but transition c is fired. Then the behaviour of the net is decoupled: any shuffle of the sequences $a_i \dots a_4 \cdot a_1 \cdot a_2 \cdot a_3 \cdot a_4 \dots$ and

$bj \dots b4 \cdot b1 \cdot b2 \cdot b3 \cdot b4 \dots$ may be fired from the marking reached by c . All markings reached project along $p1, p2, p3, p4$ on vectors in the set $\{1122, 2112, 2211, 1221\}$.

Clearly, $M_0[a1 \cdot b1 \cdot b2 \cdot a2 \cdot a3 \cdot b3 \cdot b4 \cdot a4]M_0$ is a simple cycle. Now, if we let $M_0[c]M_1$, then $M_1[a1 \cdot a2 \cdot a3 \cdot a4]M_1$ and $M_1[b1 \cdot b2 \cdot b3 \cdot b4]M_1$ are also simple cycles. Thus, simple cycles with common transitions may have different Parikh vectors. The two simple cycles from M_1 induce a basis of semipositive T-invariants, but the unique cycle from M_0 cannot be decomposed to a sequence of two cycles with Parikh vectors in this basis. And of course, the cyclic sequences $a1 \cdot a2 \cdot a3 \cdot a4$ and $b1 \cdot b2 \cdot b3 \cdot b4$ cannot be fired from M_0 .

However, bounded persistent nets are also amenable to some algebraic decomposition of cycles or more exactly of Parikh vectors of cycles, as we show now.

Lemma 9. BOUNDED PERSISTENT NETS HAVE HOME MARKINGS

Let N , with some initial marking, be bounded and persistent.

There exists a marking $\widetilde{M}$ such that, for any reachable marking M , $M[\xi_M]\widetilde{M}$ for some transition sequence ξ_M .

Proof: As a bounded net, N has a finite set of reachable markings $\{M_0, \dots, M_m\}$. Put $\widetilde{M}_0 = M_0$. Proceeding inductively, select for each i from 1 up to m some marking $\widetilde{M}_i$ reachable from $\widetilde{M}_{i-1}$ and M_i , which exists by Corollary 1. Then put $\widetilde{M} = \widetilde{M}_m$. $\square$

Lemma 10. CYCLES MAY BE PUSHED TO HOME MARKINGS

Let N , with some initial marking, be bounded and persistent, and let $\widetilde{M}$ be a home marking.

For any cycle $M[\rho]M$, there exists a cycle $\widetilde{M}[\widetilde{\rho}]\widetilde{M}$ such that $\Psi(\rho) = \Psi(\widetilde{\rho})$.

Proof: Let $M[\xi]\widetilde{M}$. One applies Lemma 2 in M with $\gamma = \xi$ and $\kappa = \rho$. $\square$

Theorem 3. SECOND DECOMPOSITION THEOREM

Let N , with some initial marking, be bounded and persistent. There exists a reachable marking $\widetilde{M}$ and a finite set of transition-disjoint cycles $\widetilde{M}[\rho_i]\widetilde{M}$ ($1 \leq i \leq n$) such that for any reachable marking M and for any cycle $M[\rho]M$, the Parikh vector of this cycle decomposes as $\Psi(\rho) = \sum_{i=1}^n n_i \times \Psi(\rho_i)$ for some non-negative integral numbers n_i .

Proof: By Lemma 9, N has at least one home marking $\widetilde{M}$. By Lemma 10, the set of Parikh vectors of cycles of $N = (S, T, F, M_0)$ coincides with the set of Parikh vectors of cycles of the bounded persistent and reversible net $\widetilde{N} = (S, T, F, \widetilde{M})$. The theorem follows by direct application of Theorem 2 to the net $\widetilde{N}$. $\square$

In the rest of the section, we propose an independent proof of Theorem 3, relying on Keller's theorem and on Lemmas 1, 2, 4 and 9, and entailing all results from section 5, including Theorem 2. We first strengthen the notion of simplicity.

Definition 2. HYPERSIMPLICITY

A cycle $M[\rho]M$ is called hypersimple if the Parikh vector $\Psi(\rho)$ differs from $\Psi(\rho_1) + \Psi(\rho_2)$ for any two non-trivial cycles $M_1[\rho_1]M_1$ and $M_2[\rho_2]M_2$ from reachable markings M_1 and M_2 . $\square$

A hypersimple cycle is always simple, but the converse is not true (as shown by the example presented at the beginning of the section).

Lemma 11. IN THE REVERSIBLE CASE, EVERY SIMPLE CYCLE IS HYPERSIMPLE

Let N , with some initial marking, be bounded, reversible, and persistent. Let M be a reachable marking and let $M[\rho]M$ be a simple cycle.

Then $M[\rho]M$ is hypersimple.

Proof: Suppose for a contradiction that $\Psi(\rho) = \Psi(\rho_1) + \Psi(\rho_2)$ for non-trivial cycles $M_1[\rho_1]M_1$ and $M_2[\rho_2]M_2$ from reachable markings M_1 and M_2 . By reversibility, $M_1[\xi]M_0$ for some transition sequence ξ . Let $M_0[\chi]M$. By Lemma 2 applied in M_1 with $\kappa = \rho_1$ and $\gamma = \xi\chi$, $\Psi(\rho_1) = \Psi(\rho'_1)$ for some cycle $M[\rho'_1]M$. By definition of the residual operation, $\rho'_1 \dot{-} \rho = \varepsilon$ because $\Psi(\rho'_1)$ is smaller than $\Psi(\rho)$. By Keller's theorem applied to $M[\rho]M$ and $M[\rho'_1]M$, $M[\rho'_1 \dot{-} \rho]M'$ and $M[\rho \dot{-} \rho'_1]M'$ for some M' , with $\rho(\rho'_1 \dot{-} \rho) \equiv_M \rho'_1(\rho \dot{-} \rho'_1)$. As $\rho'_1 \dot{-} \rho = \varepsilon$, $M = M'$ and $\Psi(\rho) = \Psi(\rho'_1) + \Psi(\rho \dot{-} \rho'_1)$. Recalling that $\Psi(\rho) = \Psi(\rho_1) + \Psi(\rho_2)$, we have that $\Psi(\rho'_1) = \Psi(\rho_1)$ and $\Psi(\rho \dot{-} \rho'_1) = \Psi(\rho_2)$ both differ from the null vector. Now $M[\rho'_1]M[\rho \dot{-} \rho'_1]M$, and therefore $M[\rho]M$ is not a simple cycle. $\square$

Lemma 12 below is an adaptation of Lemma 5 to the non-reversible case.

Lemma 12. HYPERSIMPLE CYCLES SHARING TRANSITIONS ARE PARIKH-EQUIVALENT

Let N , with some initial marking, be bounded and persistent. Moreover, let M, M' be reachable markings and $M[\tau]M$ and $M'[\sigma]M'$ be two hypersimple cycles. If some common transition t occurs in both cycles, i.e. $\Psi(\tau)(t) \neq 0 \neq \Psi(\sigma)(t)$, then $\Psi(\tau) = \Psi(\sigma)$.

Proof: By Corollary 1, there exists a marking M'' and two firing sequences ξ and χ such that $M[\xi]M''$ and $M'[\chi]M''$.

By Lemma 2 applied in M with $\gamma = \xi$ and $\kappa = \tau$, there exists a transition sequence τ' such that $M''[\tau']M''$ and $\Psi(\tau) = \Psi(\tau')$.

By Lemma 2 applied in M' with $\gamma = \chi$ and $\kappa = \sigma$, there exists a transition sequence σ' such that $M''[\sigma']M''$ and $\Psi(\sigma) = \Psi(\sigma')$.

Let $\tau' = \tau'_1\tau'_2$ and $\sigma' = \sigma'_1\sigma'_2$ such that t occurs neither in τ'_1 nor in σ'_1 , and let m and m' be the two markings such that $M''[\tau'_1]m$ and $M''[\sigma'_1]m'$, respectively.

By Keller's theorem, applied to $M''[\tau'_1]m$ and $M''[\sigma'_1]m'$, there exists a marking m'' such that $m[\sigma'_1 \dot{-} \tau'_1]m''$ and $m'[\tau'_1 \dot{-} \sigma'_1]m''$.

By Lemma 2 applied in m with $\gamma = \sigma'_1 \dot{-} \tau'_1$ and $\kappa = t\tau'_2\tau'_1$, there exists a transition sequence τ'' such that $m''[\tau'']m''$ and $\Psi(\tau'') = \Psi(t\tau'_2\tau'_1) = \Psi(\tau)$.

By Lemma 2 applied in m' with $\gamma = \tau'_1 \dot{-} \sigma'_1$ and $\kappa = t\sigma'_2\sigma'_1$, there exists a transition sequence σ'' such that $m''[\sigma'']m''$ and $\Psi(\sigma'') = \Psi(t\sigma'_2\sigma'_1) = \Psi(\sigma)$.

Now $m[t]$, $m[\sigma'_1 \dot{-} \tau'_1]m''$, and transition t does not occur in $\sigma'_1 \dot{-} \tau'_1$ since it does not occur in σ'_1 . By persistency, it follows that $m''[t]\tilde{m}$ for some marking $\tilde{m}$.

As $\Psi(t)$ is smaller than or equal to $\Psi(\tau'') = \Psi(\tau)$, $t \bullet \tau'' = \varepsilon$. By Keller's theorem, applied to $m''[\tau'']m''$ and $m''[t]\tilde{m}$, $\tilde{m}[\tau'' \bullet t]m''$. As the Parikh vector of the cycle $m''[t]\tilde{m}[\tau'' \bullet t]m''$ is equal to $\Psi(\tau'') = \Psi(\tau)$, this cycle is hypersimple. By Keller's theorem, applied to $m''[\sigma'']m''$ and $m''[t]\tilde{m}$, one can construct in a similar way a hypersimple cycle $m''[t]\tilde{m}[\sigma'' \bullet t]m''$. As every hypersimple cycle is simple, and both cycles start with t from m'' , Lemma 4 applies, entailing that $t(\tau'' \bullet t) \equiv_{m''} t(\sigma'' \bullet t)$ and hence that $\Psi(\tau) = \Psi(\tau'') = \Psi(\sigma'') = \Psi(\sigma)$. The claim of the lemma has thus been established. $\square$

Theorem 3 follows easily from Lemmas 9, 10 and 12. First, any cycle $M[\rho]M$ may be pushed to the chosen home marking $\tilde{M}$ and then decomposed to a sequence of simple cycles through $\tilde{M}$. Second, any simple cycle $\tilde{M}[\rho]\tilde{M}$ is hypersimple: if it was otherwise, i.e. $\Psi(\rho) = \Psi(\rho_1) + \Psi(\rho_2)$ where $M_1[\rho_1]M_1$ and $M_2[\rho_2]M_2$, then by pushing these two cycles to $\tilde{M}$, one would get a decomposition of $\tilde{M}[\rho]\tilde{M}$ which is supposed to be simple. Third, two hypersimple cycles through $\tilde{M}$ are either transition-disjoint or Parikh equivalent.

For bounded persistent and reversible nets, Lemma 5 follows from Lemmas 11 and 12. And since in a reversible net, every reachable marking is a home marking, Theorem 2 and Corollary 2 follow from Lemma 10 and Theorem 3.

For non-reversible bounded and persistent nets, the set of T-invariants realised at marking M , i.e. equal to $\Psi(\alpha)$ for some cycle $M[\alpha]M$, depends on M . By the same reasoning as in the proof of Lemma 11 (with $\rho = \alpha$ and $\rho'_1 = \beta$) one can show that for any two cycles $M[\alpha]M$ and $M[\beta]M$ from a reachable marking M , if $\Psi(\alpha)$ is greater than $\Psi(\beta)$, then $\Psi(\alpha) = \Psi(\beta) + \Psi(\gamma)$ for some cycle $M[\gamma]M$. In view of Dickson's lemma, it follows that the set of T-invariants realised at M is generated by a unique minimal integral basis $\mathcal{B}_M = \{\Psi(\rho_1), \dots, \Psi(\rho_n)\}$ where $M[\rho_i]M$ for $i = 1, \dots, n$ and moreover, all these cycles are simple and therefore transition disjoint (in view of Lemma 4). In the reversible case, the unique minimal integral basis $\mathcal{B}$ is the same for all reachable markings.

7 Some Comments

In the special case of strongly connected marked graphs, Theorems 2 and 3 coincide and are well-known. In this case, there is only one vector in $\mathcal{B}$, namely the T-invariant $\underline{1}$, that is, the T-vector containing a 1 in each entry. Every reproducing sequence is simply a multiple of this vector.

The boundedness condition on nets is essential in all our theorems. Consider Figure 5. The net shown there is persistent, but not bounded. One of its coverability graphs is shown on the right-hand side of the figure. There are two simple cycles, ca and cb , which are not transition-disjoint and do not have the same Parikh vector, and the conclusions of Theorems 2 and 3 are violated (for the infinite reachability graph).

The condition of persistency can neither be omitted nor replaced by weak persistency as defined in [10]. Consider Figure 6. The net shown there is bounded and it is not persistent, but it is weakly persistent. The reachability graph has

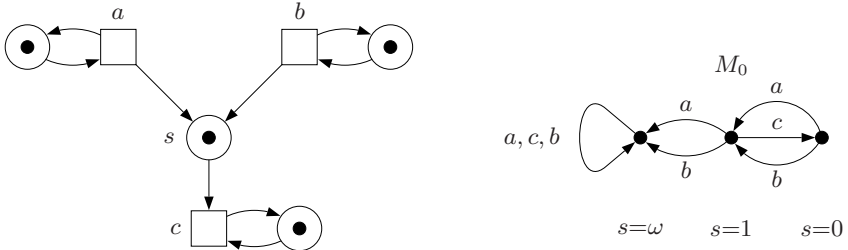


Fig. 5. A non-bounded Petri net (l.h.s.) and a coverability graph (r.h.s.)

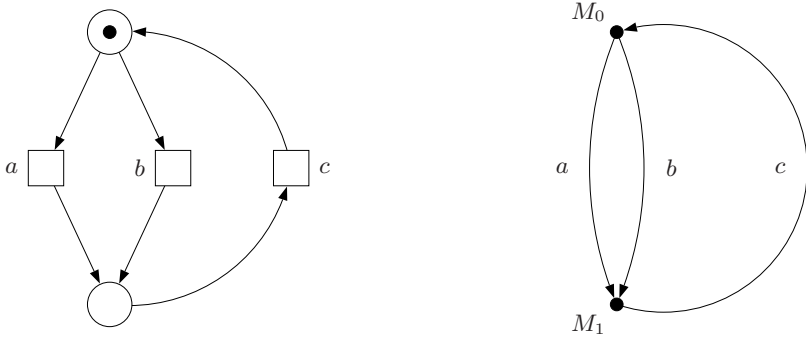


Fig. 6. A non-persistent Petri net (l.h.s.) and its reachability graph (r.h.s.)

two simple cycles from M_0 , ac and bc , which are not transition-disjoint and do not have the same Parikh vector. The conclusions of Lemma 4 and Theorems 2 and 3 are violated.

Theorem 2 indicates that, when $N = (S, T, F, M)$ is bounded, persistent and reversible, the language of this net may be written as a parallel composition $L(N) = \parallel_{i=1}^n (L_i)^\omega$ where $L_i = [\rho_i]_{\equiv_M}$ is the equivalence class of a cycle $M[\rho_i]M$ with $\rho_i \in T_i^*$ and $T_i \cap T_j = \emptyset$ for $i \neq j$. As N is persistent, $L(N)$ does not change when a place that connects different subsets T_i is replicated to as many places as connected subsets and when the flow arcs are distributed accordingly. N may therefore be decomposed to n disjoint, bounded, persistent and reversible subnets generating altogether the same language. From the results in [4], they also generate the same reachable state graph up to an isomorphism. Bounded, persistent and reversible nets are therefore amenable to modular verification. The authors hope that the results described in the present paper will also help solving two challenging problems described in [1], one relating to the conflict-freeness hierarchy defined in that paper, and another one relating to the concept of separability, also described there and investigated more fully in [2].

The first challenge is to prove or to disprove that bounded and live persistent nets can essentially be simplified to behaviourally conflict-free nets. The latter means that any two enabled transitions do not share a common input place. For

instance, the net shown in Figure 1 can trivially be simplified in this way, by omitting the middle place with two tokens.

The second challenge is to prove or to disprove that bounded and live persistent Petri nets are separable in the following sense: If the initial marking, say M , of such a net, say N , is a k -multiple of another one, then N with initial marking M behaves as k disjoint copies of N with initial marking $(1/k) \cdot M$. It is shown in [2] that general marked graphs enjoy this property, but that bounded and live free-choice nets do not. Furthermore, [1] contains an example of a non-separable, bounded, output-nonbranching net which is, however, not live.

Acknowledgments

The first author is indebted to Harro Wimmel for listening to several attempts at formulating and/or proving these results and for providing insightful suggestions. We would also like to thank Javier Esparza and Raymond Devillers for helpful comments, and an anonymous reviewer for pointing out reference [10].

References

1. Best, E., Darondeau, P., Wimmel, H.: Making Petri Nets Safe and Free of Internal Transitions. *Fundamenta Informaticae* 80, 75–90 (2007)
2. Best, E., Esparza, J., Wimmel, H., Wolf, K.: Separability in Conflict-free Petri Nets. In: Virbitskaite, I., Voronkov, A. (eds.) *PSI 2006*. LNCS, vol. 4378, pp. 1–18. Springer, Heidelberg (2007)
3. Commoner, F., Holt, A.W., Even, S., Pnueli, A.: Marked Directed Graphs. *J. Comput. Syst. Sci.* 5(5), 511–523 (1971)
4. Darondeau, P.: Equality of Languages Coincides with Isomorphism of Reachable State Graphs for Bounded and Persistent Petri Nets. *Information Processing Letters* 94, 241–245 (2005)
5. Desel, J., Esparza, J.: *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science, 242 pages (1995) ISBN:0-521-46519-2
6. Grabowski, J.: The Decidability of Persistence for Vector Addition Systems. *Information Processing Letter* 11(1), 20–23 (1980)
7. Keller, R.M.: A Fundamental Theorem of Asynchronous Parallel Computation. In: Tse-Yun, F. (ed.) *Parallel Processing*. LNCS, vol. 24, pp. 102–112. Springer, Heidelberg (1975)
8. Landweber, L.H., Robertson, E.L.: Properties of Conflict-Free and Persistent Petri Nets. *JACM* 25(3), 352–364 (1978)
9. Stark, E.W.: Concurrent Transition Systems. *TCS* 64, 221–269 (1989)
10. Yamasaki, H.: On Weak Persistency of Petri Nets. *Information Processing Letters* 13(3), 94–97 (1981)

A Proof of Theorem 1

Let N be a persistent net and let τ and σ be two firing sequences starting from some reachable marking M . We show that $\tau(\sigma \overset{\bullet}{\dashv} \tau)$ and $\sigma(\tau \overset{\bullet}{\dashv} \sigma)$ are also firing sequences from M , and that they are permutations of each other from M . Let $\tau =$

$t_1 \dots t_n$ and $\sigma = x_1 \dots x_m$. Persistency allows us to complete “small diamonds”. For instance, if $t_1 \neq x_1$, then both $M[t_1x_1]$ and $M[x_1t_1]$. The proof proceeds by “completing big diamonds”, such as the one shown in Figure 7. Special care needs to be taken if one of the t_i equals one of the x_j . We proceed systematically through $t_1, \dots, t_n$, in this order, trying to match as many transitions t_i as possible with transitions x_j from σ .

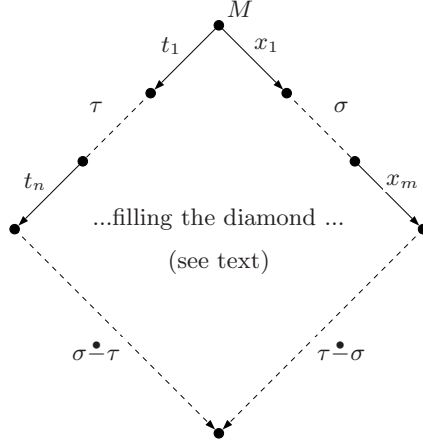


Fig. 7. Outlining the proof of Keller’s theorem

The starting point are the North, West and East corners of the diamond shown in Figure 7. For each t_i , we will define an i ’th line from Northwest to Southeast. Formally, each line is of the form $\widehat{M}[\widehat{\sigma}]$, where $\widehat{M}$ is a marking reachable from M by some subsequence of τ and $\widehat{\sigma}$ is some subsequence of σ . Line 0, i.e. the starting line, is given by σ which leads from the North corner to the East corner; formally, it is defined to be $M[\sigma]$. We distinguish two cases:

- If t_i does not have some transition $x_j = t_i$ on the previous (the $(i-1)$ ’th) line, we draw a new (i ’th) line after t_i which is an exact parallel to the previous line, and we cut the resulting parallelogram by small t_i -labelled arcs from Northeast to Southwest. This is justified by the small-diamond completion property of persistent nets.
- If, however, on the $(i-1)$ ’th line, there are some x -transitions that are the same as t_i , we choose the first j for which $x_j = t_i$ holds and we draw an exact parallel only up to the starting point of x_j . The endpoint of this parallel will be merged with the endpoint of x_j , and from that point onwards, the new line is the same as the previous line. Thus, the new line contains one arc (namely, an arc labelled with x_j) less than the previous line. If x_j happens to be the first x -transition on the previous line, this construction corresponds to the special case of merging the endpoints of t_i and x_j . The resulting parallelogramoid is again subdivided by small t_i -arcs, but this time only up to the arc before x_j , if there is any.

It is clear that the last line, from the West corner to the resulting South corner, corresponds to the sequence $\sigma \overset{\bullet}{\dashv} \tau$ of unmatched x -transitions, and it is also not hard to see that the line from the East corner to the South corner corresponds to the sequence $\tau \overset{\bullet}{\dashv} \sigma$ of unmatched t -transitions. $\square$

Figures 8 and 9 exhibit some examples with $\tau = t_1 t_2 t_3$ and $\sigma = x_1 x_2 x_3$.

In Figure 8, we assume that the only common transition between $t_1 t_2 t_3$ and $x_1 x_2 x_3$ is $t_2 = x_3$. If, additionally, $t_1 = x_1$, we get the diamond shown on the left-hand side of Figure 9. If $t_3 = x_2$ holds further, then the three southmost nodes are merged into a single node, and the t_3 - and x_2 -arcs leading into them collapse into a single arc labelled t_3 (or x_2), and we arrive at the diamond shown on the right-hand side of Figure 9.

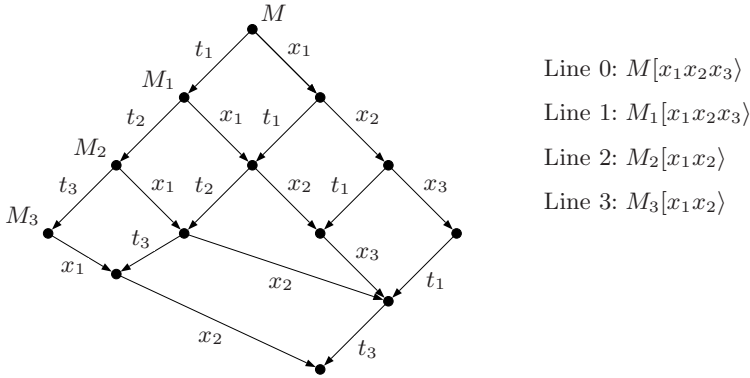


Fig. 8. Diamond for $t_1 t_2 t_3$ and $x_1 x_2 x_3$, assuming $t_2 = x_3$

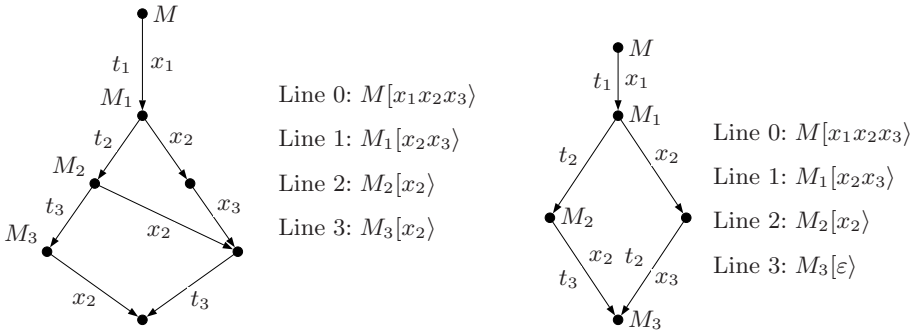


Fig. 9. Diamonds, assuming $t_1 = x_1 \wedge t_2 = x_3$ (l.h.s.) and $t_1 = x_1 \wedge t_2 = x_3 \wedge t_3 = x_2$ (r.h.s.)

Compositional Specification of Web Services Via Behavioural Equivalence of Nets: A Case Study*

Filippo Bonchi, Antonio Brogi, Sara Corfini, and Fabio Gadducci

Department of Computer Science – University of Pisa, Italy
{fibonchi,brogi,corfini,gadducci}@di.unipi.it

Abstract. Web services represent a promising technology for the development of distributed heterogeneous software systems. In this setting, a major issue is to establish whether two services can be used interchangeably in any context. This paper illustrates — through a concrete scenario from banking systems — how a suitable notion of behavioural equivalence over Petri nets can be effectively employed for checking the correctness of service specifications and the replaceability of (sub)services.

1 Introduction

Web services are emerging as a promising technology for the development of next generation distributed heterogeneous software systems. Roughly, a Web service is a self-describing software component universally accessible by means of standard protocols (WSDL, UDDI, SOAP). Platform-independence and ubiquity make Web services the building blocks for developing new complex applications [1]. In this scenario, a prominent issue is to establish whether two services are *behaviourally* equivalent, i.e., such that an external observer can not tell them apart. Yet, standard WSDL interfaces provide services with purely syntactic descriptions: they do not include information on the possible interaction between services, thus inhibiting the *a priori* verification of any behavioural property.

During the last years, various proposals have been put forward to feature more expressive service descriptions that include semantics (viz., ontology-based) and behaviour information about services. One of the major efforts in this direction is OWL-S [2], a high-level ontology-based language for describing services, proposed by the OWL-S coalition. In particular, OWL-S service descriptions include a declaration of the interaction behaviour of services (the so-called *process model*), which provides the needed information for the *a priori* analysis and verification of behavioural properties of (compositions of) services.

In this perspective we defined in [3] a suitable notion of *behavioural equivalence* for OWL-S described Web services represented by means of OCPN nets. OCPN nets (for Open Consume-Produce-Read nets) are a simple variant of the standard Condition/Event Petri nets, designed to address data persistency. In particular, an OCPN net is equipped with two disjoint sets of places, namely, *control* and

* Research partially supported by the EU FP6-IST IP 16004 SENSORIA and STREP 0333563 SMEPP, and the MIUR FIRB TOCAI.IT and PRIN 2005015824 ART.

data places, to naturally model the control flow and the data flow of a Web service, and with an *interface*, which establishes those data places that can be observed externally. The main features of the equivalence presented in [3], named *saturated bisimilarity*, are *weakness*, as it equates externally indistinguishable services by abstracting from the number of internal steps, and *compositionality*, as it is also a congruence. Furthermore, the equivalence was proved there to be *decidable*, by characterizing it in terms of an alternative, clearly decidable behavioural equivalence, so-called *semi-saturated* bisimilarity.

This paper focuses on exploiting the behavioural equivalence introduced in [3] in order to outline a methodology for addressing two specific issues related to service specification. Namely, for checking whether a service specification is equivalent to a service implementation, and whether a (sub)service may replace another (sub)service without altering the behaviour of the whole application. In doing so, a simpler decidable characterization of saturated bisimilarity is introduced, which is based on the standard notion of *weak* bisimilarity.

The methodology is presented by directly instantiating it on a concrete example scenario, the finance case study of the SENSORIA project, consisting of a banking service which grants loans to bank customers. In the first scenario we present, we detail the complete behaviour of the banking service, and we propose a possible specification to externally publish it. We employ weak bisimilarity to check whether the proposed specification properly describes the concrete banking service implementation, i.e., to verify whether the externally observable behaviour of the service implementation and of the banking service specification are equivalent. In the second scenario, we consider the specific part of the banking service which evaluates the customer rating. We present two services with different behaviour, yet both computing customer ratings. By applying weak bisimilarity, we verify whether these two latter services are equivalent as well as whether they can replace part of the banking service affecting neither the internal behaviour of the banking service nor its public interface.

The rest of the paper is organized as follows. Section 2 illustrates the finance case study together with a brief review of OWL-S. The first part of Section 3 briefly recalls the results of [3] by discussing OCPR nets and saturated bisimilarity; the second part introduces weak bisimilarity, and it presents a formal encoding of OWL-S service descriptions into OCPR nets, and a new compositionality result for these nets. Section 4 exploits these results on compositional specification for outlining a methodology addressing the issues on service specification mentioned above. In Section 5 the methodology is then instantiated on the finance case study described in Section 2. Finally, we discuss related work and we draw some concluding remarks in Section 6.

2 Case Study: A Credit Scenario for the Banking System

In order to provide a proper motivation for our proposals, this section illustrates an example scenario concerning banking systems. After an informal outline, the scenario is specified in the OWL-S description language. The section is

then rounded up by discussing some issues concerning service replaceability (i.e., dynamic reconfigurations) and service publication (i.e., alternative user views).

First of all, though, we consider important to observe that the banking scenario is inspired by the finance case study described by S&N AG netBank solutions (<http://www.s-und-n.de/>) which is one of the enterprises involved in the SENSORIA project (<http://www.sensoria-ist.eu/>). In particular, CreditPortal is a Web service that grants loans to bank customers. CreditPortal implements three steps, namely, (1) authentication of the customer and upload of her/his personal data, (2) evaluation of the customer credit request, and (3) formulation of the loan offer. In the first step, after logging into the bank system, the customer has to upload information regarding balance and offered guarantees. In the second step, CreditPortal evaluates the customer reliability and computes a rating of the credit request. Finally, in the last step CreditPortal either decides to grant the loan to the customer and to build an offer, or it rejects the credit request.

A short recap on OWL-S

As anticipated in the Introduction, we consider Web services specified in OWL-S [2], an ontology-based language for semantically describing services. In particular, an OWL-S service is advertised by three different files, namely the *service profile*, describing the functional (i.e., inputs/outputs) and extra-functional attributes of the service, the *process model* detailing the service behaviour in terms of its component processes, and the *service grounding*, explaining how to access the service by specifying protocol and message format information.

In the rest of the paper we point our attention to the OWL-S process model, as we focus on service behaviour. The process model describes a service as a composite or an atomic process. An atomic process can not be decomposed further and it executes in a single step, while a composite process is built up by using control constructs: **sequence** (sequential execution), **if-then-else** (conditional execution), **choice** (non-deterministic execution), **split** (parallel execution), **split+join** (parallel execution with synchronization), **any-order** (unordered sequential execution), **repeat-while** and **repeat-until** (iterative execution).

Specifying the scenario using the OWL-S process model

The OWL-S specification of CreditPortal is available at [4]: for the convenience of the reader, Fig. 1 shows a compact tree representation of the CreditPortal process model. Each internal node is labelled with the type of the composite process it represents, and, in case of conditional and iterative control constructs, also with a condition. Each leaf is associated with the inputs and the outputs of the corresponding atomic process. It is worth noting that in the OWL-S description of CreditPortal each input and output parameter is annotated with a concept defined in a shared ontology. As depicted in Fig. 1, the CreditPortal process model is a sequence process whose left-most child is a repeat-until construct representing the customer authentication phase. The customer may either log in with an existing account (**login**) or create a new account (**createAccount**) until either the log in to the system is successful (*validData* = true), or the system rejects the login definitively (*rejectedLogin* = true). Next, if the customer did not provide a valid login, CreditPortal terminates (**invalidLogin**). Otherwise, it asks the customer for

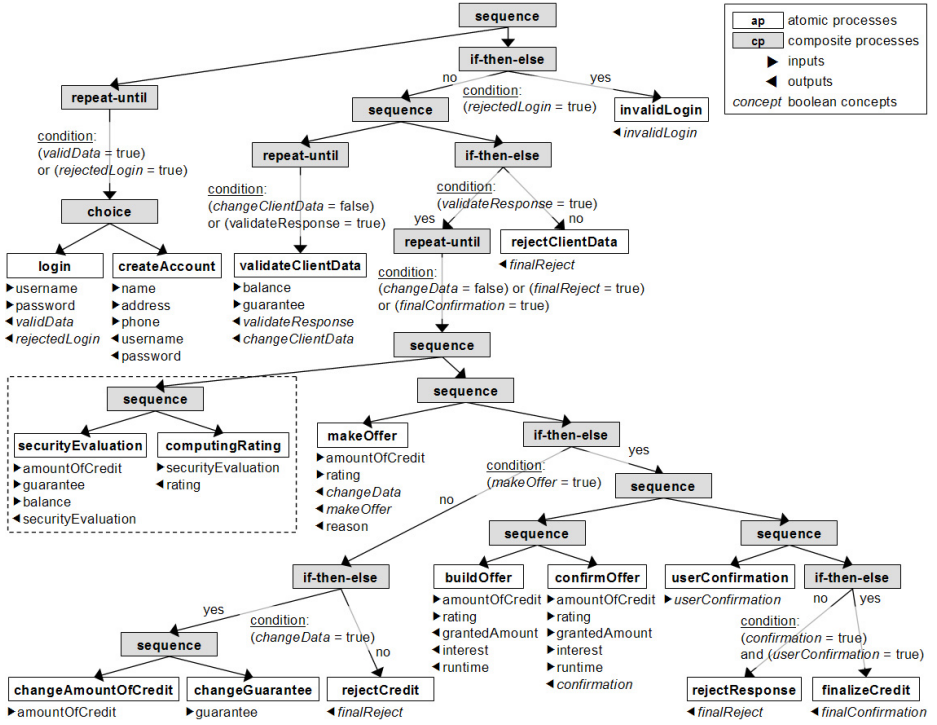


Fig. 1. The OWL-S process model of the CreditPortal service

the personal financial data (`validateClientData`) until either a valid information is uploaded (`validateResponse = true`) or the system rejects the credit request (`changeClientData = false`). Then, if the customer did not provide valid financial data, CreditPortal terminates (`rejectClientData`), otherwise the customer credit request evaluation phase starts. CreditPortal, taking into account the requested amount of credit, evaluates the customer security (`securityEvaluation`), computes the customer rating (`computingRating`) and decides whether or not to make an offer to the customer (`makeOffer`). If so (`makeOffer = true`), it builds the offer (`buildOffer`), formally confirms the offer (`confirmOffer`) and asks the customer for a final confirmation (`userConfirmation`). If CreditPortal and the customer agree on the offer (`confirmation = true` and `userConfirmation = true`), the offer is finalized (`finalizeCredit`), otherwise CreditPortal rejects the credit request (`rejectResponse`). Instead, if CreditPortal does not want to make an offer (`makeOffer = false`), it can either reject the credit request (`rejectCredit`) or allow the customer to update the financial data (`changeAmountOfCredit` and `changeGuarantee`). In the latter case, the evaluation phase is repeated.

An outline of two issues in service composition

Let us now discuss two issues concerning service publication and service replaceability. The process model presented in Fig. 1 describes the full behaviour of the

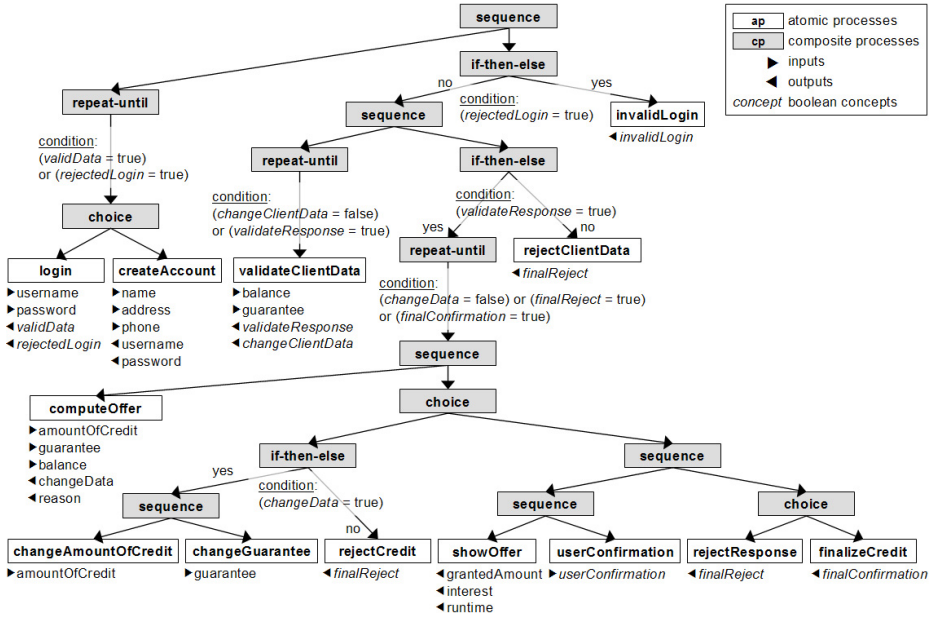


Fig. 2. Public specification of the CreditPortal service

CreditPortal service. Yet, it is reasonable that the CreditPortal provider wants to publish a simpler specification of the service by hiding unnecessary and/or confidential details of its implementation. For instance, Fig. 2 depicts a possible public specification of CreditPortal, which hides several internal parameters and operations. A methodology to check whether the public specification properly advertises the internal behaviour of a service is hence required.

Consider now the dotted section of the service specification represented in Fig. 1. It consists of a sequence of two atomic processes, i.e., securityEvaluation and computingRating, that takes as input the requested amount of credit, the balance and the provided guarantees of a customer, and then evaluates the reliability and the rating. Let us now suppose that the CreditPortal provider (i.e., the bank) wants to enhance its service and hence decides to externalize the CreditPortal section which computes customer rating, viz., the dotted area of Fig. 1. For instance, suppose that two services which compute customer rating are available, that is, RatingOne and RatingTwo whose OWL-S process models are illustrated in Figs. 3 and 4, respectively. More precisely, RatingOne firstly computes three separate evaluations of the customer and then it returns an average rating, RatingTwo computes the customer rating and only if necessary (e.g., if the first rating exceeds a threshold value) it performs a second and possibly a third evaluation of the customer. RatingTwo may be more convenient for the bank, as it does not always compute three separate and expensive customer evaluations, yet, RatingOne provides a more accurate customer evaluation. In both cases, the bank needs to verify whether the dotted area of CreditPortal in Fig. 1 can be

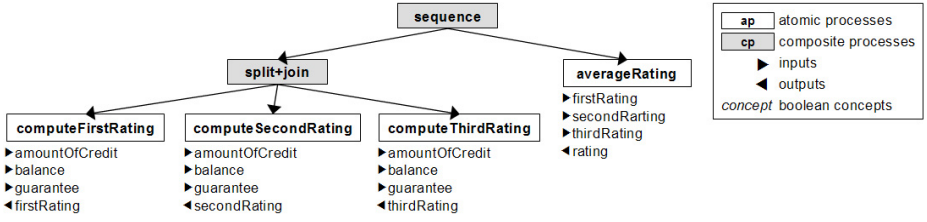


Fig. 3. The OWL-S process model of the RatingOne service

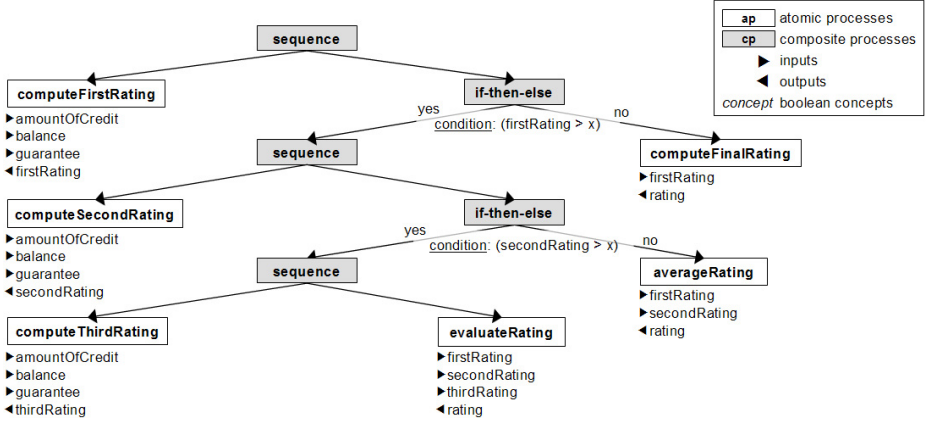


Fig. 4. The OWL-S process model of the RatingTwo service

replaced with RatingOne or RatingTwo affecting neither the internal behaviour of CreditPortal nor the correctness of its public specification.

3 Formal Reasoning on Service Behaviour

In [3] we introduced *Open Consume-Produce-Read* (OCPR) nets – a variant of standard Petri nets [5] – to naturally model the behaviour of (OWL-S described) services. In Section 3.1 we recall basic definitions and results on OCPR nets from [3], Section 3.2 introduces the decidable weak bisimilarity, while Section 3.3 shows a formal encoding of OWL-S into OCPR nets, and presents a simple characterisation result concerning place hiding.

3.1 Open Consume-Produce-Read Nets

We recall CPR and Open CPR nets, namely, CPR nets that can interact with the environment (i.e., the contexts) through an interface; and we show the behavioural congruence introduced in [3] for discussing (sub)service replaceability.

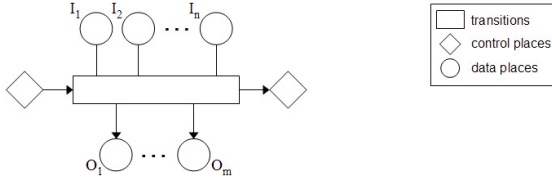


Fig. 5. Modelling atomic operations as CPR net transitions

Consume-Produce-Read nets

A CPR net is equipped with two disjoint sets of places: the *control* and the *data* places which respectively model the control and the data flow of a Web service. Besides the textual presentation, the graphical notation is depicted in Fig. 5.

Definition 1 (CPR net). A consume-produce-read net (simply, CPR net) N is a five-tuple $(CP_N, DP_N, T_N, CF_N, DF_N)$ where

- CP_N is a finite set of control places,
- DP_N is a finite set of data places (disjoint from CP_N),
- T_N is a finite set of transitions,
- $CF_N \subseteq (CP_N \times T_N) \cup (T_N \times CP_N)$ is the control flow relation,
- $DF_N \subseteq (DP_N \times T_N) \cup (T_N \times DP_N)$ is the data flow relation.

A marking M for N is a finite set of places in $P_N = CP_N \cup DP_N$.

As for standard nets, we associate a *pre-set* and a *post-set* with each transition t , together with two additional sets, called *read-set* and *produce-set*.

Definition 2 (pre-, post-, read-, and produce-set). Given a CPR net N , we define for each $t \in T_N$ the sets

$$\begin{aligned} {}^\diamond t &= \{s \in CP_N \mid (s, t) \in CF_N\} & t^\diamond &= \{s \in CP_N \mid (t, s) \in CF_N\} \\ {}^\bullet t &= \{s \in DP_N \mid (s, t) \in DF_N\} & t^\bullet &= \{s \in DP_N \mid (t, s) \in DF_N\} \end{aligned}$$

which denote respectively the pre-set, post-set, read-set and produce-set of t .

Definition 3 (firing step). Let N be a CPR net. Given a transition $t \in T_N$ and a marking M for N , a firing step is a triple $M[t]M'$ such that $({}^\diamond t \cup {}^\bullet t) \subseteq M$ and $(M \cap t^\diamond) \subseteq {}^\diamond t$ (M enables t), and moreover $M' = (M \setminus {}^\diamond t \cup t^\diamond \cup t^\bullet)$.

We write $M \setminus M'$ if there exists some t such that $M[t]M'$.

The enabling condition states that the tokens of the pre-/read-set of a transition have to be contained in the marking, and that the marking does not contain any token in the post-set of the transition, unless it is consumed and regenerated (as for C/E nets). Note that data places act instead as sinks, that is, the occurrence of a token may be checked (read), but the token is never consumed, nor it may disable a transition. This is coherent with our underlying modelling choice with respect to Web services, argued in [6], where the persistency of data is assumed: once it is produced, a data remains always available.

Open CPR nets and CPR contexts

The first step for defining compositionality is to equip nets with an *interface*.

Definition 4 (Open CPR net). Let N be a CPR net. An interface for N is a triple $\langle i, f, OD \rangle$ such that $i \neq f$ and

- i is a control place (i.e., $i \in CP_N$), the initial place;
- f is a control place (i.e., $f \in CP_N$), the final place; and
- OD is a set of data places (i.e., $OD \subseteq DP_N$), the open data places.

An interface is an *outer interface* O for N if there exists no transition $t \in T_N$ such that either $i \in t^\circ$ or $f \in {}^\circ t$. An *open CPR net* $\mathcal{N}$ (OCPR for short) is a pair $\langle N, O \rangle$, for N a CPR net and O an outer interface for N .

Given an open net $\mathcal{N}$, $Op(\mathcal{N})$ denotes the set of *open places*, which consists of those places occurring in the interface, initial and final places included. Furthermore, the places of $\mathcal{N}$ not belonging to $Op(\mathcal{N})$ constitute the *closed places*.

The graphical notation used to represent OCPR nets can be observed, e.g., in Fig. 9. The bounding box of the OCPR net there represents the outer interface of the net: the initial and final control places are going to be used to compose the control of services, and the open data places to share data.

Next, we symmetrically define an *inner interface* for N as an interface such that there is no transition $t \in T_N$ satisfying either $f \in t^\circ$ or $i \in {}^\circ t$.

Definition 5 (CPR context). A CPR context $C[-]$ is a triple $\langle N, O, I \rangle$ such that N is a CPR net, I and O are an inner and an outer interface for N , respectively, and $i_I \neq f_O$, $i_O \neq f_I$.

Contexts represent environments in which services can be plugged-in, i.e., possible ways they can be used by other services. Graphically speaking, as one can note in Fig. 6, a context is an open net with an hole, represented by a gray area. The border of the hole denotes the inner interface of the context, while the bounding box is the outer interface. An OCPR net can be inserted in a context if the net outer interface and the context inner interface coincide.

Definition 6 (CPR composition). Let $\mathcal{N} = \langle N, O \rangle$ be an OCPR net and $C[-] = \langle N_C, O_C, I_C \rangle$ a CPR context, such that $O = I_C$. Then, the composite net $C[\mathcal{N}]$ is the OCPR net $(CP_N \uplus_O CP_{N_C}, DP_N \uplus_O DP_{N_C}, T_N \uplus T_{N_C}, CF_N \uplus CF_{N_C}, DF_N \uplus DF_{N_C})$ with outer interface O_C .

In other words, the disjoint union of the two nets is performed, except for those places occurring in O , which are coalesced: this is denoted by the symbol $\uplus_O$. Moreover, O_C becomes the set of open places of the resulting net.

Saturated bisimilarity for OCPR nets

Let $\mathcal{P}$ be the set of all OCPR nets with markings and let $Obs(\mathcal{N}, M) = Op(\mathcal{N}) \cap M$ be the observation made on the net $\mathcal{N}$ with marking M . Let $\rightarrow_{\mathcal{N}}$ be the reflexive and transitive closure of the firing relation $[\cdot]$ of the net N of $\mathcal{N}$.

Definition 7 (saturated bisimulation). A symmetric relation $\mathfrak{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a saturated bisimulation if whenever $(\mathcal{N}, M) \mathfrak{R} (\mathcal{N}', M')$ then

- $O_{\mathcal{N}} = O_{\mathcal{N}'}$ and $Obs(\mathcal{N}, M) = Obs(\mathcal{N}', M')$, and
- $\forall C[-] : M \rightarrow_{C[\mathcal{N}]} M_1$ implies $M' \rightarrow_{C[\mathcal{N}']} M'_1$ and $(C[\mathcal{N}], M_1) \mathfrak{R} (C[\mathcal{N}'], M'_1)$.

The union of all saturated bisimulations is called saturated bisimilarity ($\approx_S$).

Proposition 1. $\approx_S$ is the largest bisimulation that is also a congruence.

The above proposition ensures the compositionality of the equivalence, hence, the possibility of replacing one service by an equivalent one without changing the behaviour of the whole composite service. Moreover, the equivalence is “weak” in the sense that, differently from most of the current proposals, no explicit occurrence of a transition is observed. The previous definition leads to the following notion of equivalence between OCPN nets, hence, between services.

Definition 8 (bisimilar nets). Let $\mathcal{N}, \mathcal{N}'$ be OCPN nets. They are bisimilar, denoted by $\mathcal{N} \approx \mathcal{N}'$, if $(\mathcal{N}, \emptyset) \approx_S (\mathcal{N}', \emptyset)$.

Note that the above definition implies that $(\mathcal{N}, M) \approx_S (\mathcal{N}', M)$ for all M markings over open places. The negative side of $\approx_S$ is that this equivalence seems hard to be automatically decided, due to the quantification over all possible contexts. Building upon the results in [7], the main contribution of [3] is the introduction of a labeled transition system that finitely describes the interactions of a net with the environment, and such that bisimilarity on this finite transition system coincides with saturated bisimilarity, and thus it can be automatically checked. The present work further adds an alternative characterization of saturated bisimilarity via the standard notion of weak bisimilarity.

3.2 An Equivalent Decidable Bisimilarity

Saturated bisimulation seems conceptually the right notion, as it is argued in [3]. However, it also seems hard to analyze (or automatically verify), due to the universal quantification over contexts. In this section we introduce weak bisimilarity, based on a simple *labelled transition system* (LTS) distilled from the firing semantics of an OCPN net. This result improves on the characterization based on *semi-saturation* proposed in [3], since it relies on a more standard notion.

The introduction of a LTS is inspired to the theory of *reactive systems* [8]. This meta-theory suggests guidelines for deriving a LTS from an unlabelled one, choosing a set of labels with suitable requirements of minimality. In the setting of OCPN nets, the reduction relation is given by $\downarrow$, and a firing is allowed if all the preconditions of a transition are satisfied. Thus, intuitively, the minimal context that allows a firing just adds the tokens needed for that firing.

Definition 9 (labelled transition system). Let $\mathcal{N}$ be an OCPN net and $\Lambda = \{\tau\} \cup (\{+\} \times P_N) \cup (\{-\} \times CP_N)$ a set of labels, ranged over by l . The transition relation for $\mathcal{N}$ is the relation R_N inductively generated by the set of inference rules below:

$$\frac{o \in Op(\mathcal{N}) \setminus (M \cup \{f\})}{M \xrightarrow{+o}_{\mathcal{N}} M \cup \{o\}} \quad \frac{f \in M}{M \xrightarrow{-f}_{\mathcal{N}} M \setminus \{f\}} \quad \frac{M \downarrow M'}{M \xrightarrow{\tau}_{\mathcal{N}} M'}$$

where $M \xrightarrow{l}_{\mathcal{N}} M'$ means that $\langle M, l, M' \rangle \in R_{\mathcal{N}}$, and i, f denote the initial and final place of $\mathcal{N}$, respectively.

Thus, a context may add tokens in open places, as represented by the transition $\xrightarrow{+o}_{\mathcal{N}}$, in order to perform a firing. Similarly, a context may consume tokens from the final place f . A context cannot interact with the net in other ways, since the initial place i can be used by the context only as a post condition and the other open places are data places whose tokens can be read but not consumed. Instead, τ transitions represent internal firing steps, i.e., steps needing no additional token from the environment.

The theory of reactive systems ensures that, for a suitable choice of labels, the (strong) bisimilarity on the derived LTS is a congruence [8]. However, often such a bisimilarity does not coincide with the saturated one. In the case at hand, we introduce a notion of weak bisimilarity, abstracting away from the number of steps performed by nets, that indeed coincides with the saturated one.

Hereafter we use $\xrightarrow{\tau}_{\mathcal{N}}^*$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}_{\mathcal{N}}$.

Definition 10 (weak bisimulation). *A symmetric relation $\mathfrak{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if whenever $(\mathcal{N}, M) \mathfrak{R} (\mathcal{N}', M')$ then*

- $O_{\mathcal{N}} = O_{\mathcal{N}'}$ and $Obs(\mathcal{N}, M) = Obs(\mathcal{N}', M')$,
- $M \xrightarrow{+o}_{\mathcal{N}} M_1$ implies $M' \xrightarrow{+o}_{\mathcal{N}'} M'_1 \ \& \ (\mathcal{N}, M_1) \mathfrak{R} (\mathcal{N}', M'_1)$,
- $M \xrightarrow{-f}_{\mathcal{N}} M_1$ implies $M' \xrightarrow{-f}_{\mathcal{N}'} M'_1 \ \& \ (\mathcal{N}, M_1) \mathfrak{R} (\mathcal{N}', M'_1)$, and
- $M \xrightarrow{\tau}_{\mathcal{N}} M_1$ implies $M' \xrightarrow{\tau}_{\mathcal{N}'}^* M'_1 \ \& \ (\mathcal{N}, M_1) \mathfrak{R} (\mathcal{N}', M'_1)$.

The union of all weak bisimulations is called weak bisimilarity ($\approx_W$).

Theorem 1. $\approx_S = \approx_W$.

Thus, in order to prove that two OCPR nets are bisimilar, it suffices to exhibit a weak bisimulation between the states of the two nets that includes the pair of empty markings. Most importantly, though, this verification can be automatically performed, since the set of possible states of an OCPR net are finite. Hence, the result below immediately follows.

Corollary 1. $\approx_S$ is decidable.

3.3 A Compositional Encoding for OWL-S

This section presents the OCPR encoding for OWL-S service descriptions. To this aim, it introduces the notion of binary contexts, and uses it for modelling composite services. The section is rounded up by a simple result on *hiding*.

On binary contexts

As depicted in Fig. 5, an atomic process is encoded in a single transition net. Instead, the encoding of a composite service requires to extend the notion of interface, in order to accommodate the plugging of two nets into a context.

Definition 11 (binary contexts). A CPR binary context $C[-1, -2]$ is a 4-tuple $\langle N, O, I_1, I_2 \rangle$ such that the triples $\langle N, O, I_1 \rangle$, $\langle N, O, I_2 \rangle$ are CPR contexts, and $\{i_{I_1}, f_{I_1}\} \cap \{i_{I_2}, f_{I_2}\} = \emptyset$

Since it suffices for our purposes, we restrict our attention to binary contexts, the general definition being easily retrieved. Note that the control places of the inner interfaces are all different, while no condition is required for data places.

Definition 12 (binary composition). Let $\mathcal{N}_1 = \langle N_1, O_1 \rangle$, $\mathcal{N}_2 = \langle N_2, O_2 \rangle$ be OCPR nets, and $C[-1, -2]$ a CPR binary context, such that $O_1 = I_1$ and $O_2 = I_2$. Then, the composite net $C[\mathcal{N}_1, \mathcal{N}_2]$ is the OCPR net $(CP_{N_1} \uplus_U CP_{N_2} \uplus_U CP_N, DP_{N_1} \uplus_U DP_{N_2} \uplus_U DP_N, T_{N_1} \uplus T_{N_2} \uplus T_N, CF_{N_1} \uplus CF_{N_2} \uplus CF_N, DF_{N_1} \uplus DF_{N_2} \uplus DF_N)$ with outer interface O_C .

As for the unary contexts, the disjoint union of the three nets is performed, except for coalescing those places occurring in either O_1 or O_2 (denoted by $\uplus_U$).

Presenting the encoding

First we define the extension of a context for a set of data places.

Definition 13 (context extension). Let $C[-] = \langle N, O, I \rangle$ be a context. The context extension $C_A[-]$ is the context with net $N_A = (CP_N, DP_N \uplus_U A, T_N, CF_N, DF_N)$, with inner interface $I \cup A$ and outer interface $O \cup A$.

Data places are obtained by disjoint union, except for coalescing those places occurring in either O or I (denoted by $\uplus_U$). An analogous operation is defined for binary contexts. As an example, consider the context $choice_A[-1, -2]$ illustrated in Fig. 6 for $A = \{A_1, \dots, A_n\}$. It is the extension of the $choice[-1, -2]$ context (not depicted here) that just contains four transitions and six control places.

In order to define the encoding, for each OWL-S operator $\mathbf{op}$ we define a corresponding (possibly binary) context $op[-]$. Fig. 6 illustrates the encoding for all the operators. In particular, the first row shows the encoding for the three operators, namely **choice**, **sequence** and **repeat-until**, that are pivotal in the implementation of our case study. Note that the contexts depicted in Fig. 6 are the extensions $op_A[-]$ of contexts $op[-]$ corresponding to $\mathbf{op}$.

Now we can give the formal encoding. Let S be an OWL-S process model and let A be a set of data places, containing all the data occurring in S . The encoding of S with A open places is inductively defined as follows

$$\|S\|_A = \begin{cases} \mathcal{N}_{S,A} & \text{if } S \text{ is atomic,} \\ op_A[\|S_1\|_A] & \text{if } S = \mathbf{op}(S_1), \\ op_A[\|S_1\|_A, \|S_2\|_A] & \text{if } S = \mathbf{op}(S_1, S_2), \end{cases}$$

where $\mathcal{N}_{S,A}$ is the OCPR net with a single transition that reads all the input data of the atomic service S , and produces all the output data of S (as illustrated by Fig. 5), extended with all the data places of A and, as mentioned above, op_A denotes a (possibly binary) context corresponding to the OWL-S operator $\mathbf{op}$ extended with A open data places.

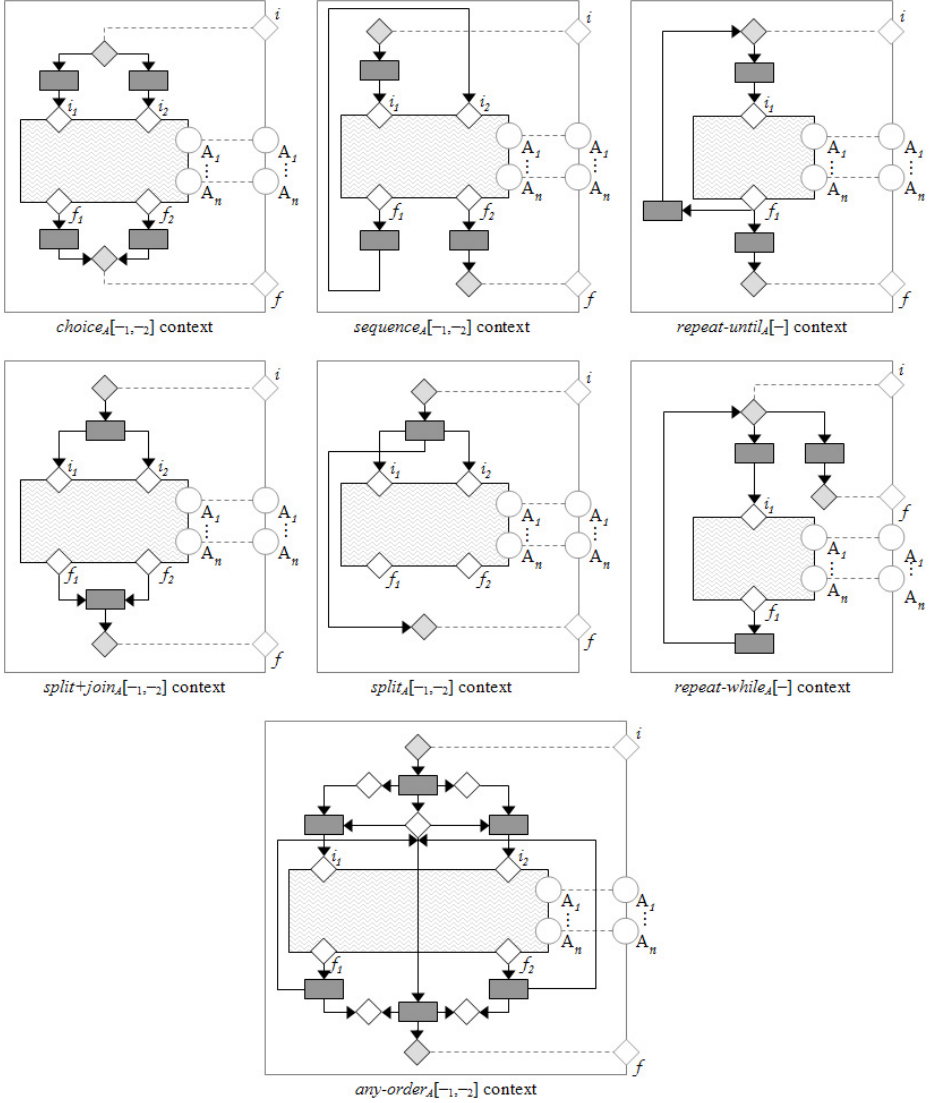


Fig. 6. Contexts corresponding to OWL-S operators extended for $A = \{A_1, \dots, A_n\}$

For the sake of simplicity, we left implicit the possible renaming of control places, needed for the composition of nets and contexts to be well-defined.

Note that the translation can be made automatic: a prototypical tool, translating OWL-S service descriptions into OCPR nets (described by a XML file) has been recently presented in [9].

With respect to our case study, it is worth noting that the encoding of conditional execution, viz., **if-then-else**, coincides with the encoding of non-deterministic execution, viz., **choice**, since our implementation of the process

model abstracts away from boolean values, as tokens have no associated value. Similar considerations hold for the operators **repeat-until** and **repeat-while**, namely, for iterative executions.

Hiding data places

The encoding presented above maps an OWL-S service into an OCPR net, where all the data places are open, i.e., they belong to the interface. As we are going to see later, this choice roughly corresponds to an orchestration view of the service, where all the available data are known. The proposition below will turn out of use for those cases where it might be necessary to abstract away from irrelevant/confidential data.

Definition 14 (hiding operator). *Let O be an outer interface and A a set of data places such that $A \subseteq O$. The hiding operator (with respect to A and O) $\nu_{A,O}$ is the context with no transitions, with inner interface O and outer interface $O \setminus A$.*

We round up the section with a simple result on disjoint compositionality, which is needed later on when discussing about (sub)service replaceability. For the sake of readability, in the following we omit the second index of an hiding operator, whenever it is clear from the context.

Proposition 2. *Let $\mathcal{N}$ be a net, $A \subseteq Op(\mathcal{N})$ a set of data places, and $C[-]$ a context such that $O_{C[-]} \cap A = \emptyset$. If the composite $C[\nu_A[\mathcal{N}]]$ is well-defined (i.e., if $I_{C[-]} = Op(\mathcal{N}) \setminus A$), then $\nu_A[C_A[\mathcal{N}]] = C[\nu_A[\mathcal{N}]]$.*

In plain terms, removing the places in A from the interface of a net $\mathcal{N}$, and then inserting the resulting net in a context $C[-]$, has the same effect as inserting $\mathcal{N}$ in a slightly enlarged context $C_A[-]$, and later on removing those same places.

4 Net Bisimulation for Publication and Replaceability

Section 3 provide us with the tools which are needed for addressing the methodology concerning service publication and replaceability discussed in Section 2.

On service publication

Let us consider an OWL-S process description S , with D_S the set of data occurring in S . The associated OCPR nets $\|S\|_{D_S}$ gives a faithful, abstract representation of the whole behaviour of the service. To check if a service and its public specification coincide, it would then suffice to simply check the equivalence of the associated nets. However, it might well happen that the service provider does not want to make all the details available to an external customer, and thus wants to hide some of the data places. This is performed by simply providing the set of data places $X \subseteq D_S$, corresponding to the data occurring in S that should be hidden, and consider $\nu_X[\|S\|_{D_S}]$. Any net (even a much simpler one) equivalent to $\nu_X[\|S\|_{D_S}]$ represents a public specification of the service.

On service replaceability

Let us consider an OWL-S process description S and its public specification P and suppose that we need to replace a subservice of S , called R , with a new service T . We must verify that, after the replacement, the external behaviour of the overall system remains the same.

Let D_S , D_P , D_R and D_T the sets of data occurring in, respectively, the descriptions S , P , R and T . Formally, “being R a subservice of S ” means that $D_R \subseteq D_S$ and that there exists a context $C[-]$ such that $C[\|R\|_{D_R}] = \|S\|_{D_S}$. Since $\approx_S$ is a congruence, it would then suffice to check that $\|R\|_{D_R} \approx_S \|T\|_{D_T}$ in order to be sure that R and T are interchangeable.

However, this condition is too restrictive, since it would imply that $D_R = D_T$. Suppose instead that T produces some data that neither R nor S produce. Or, viceversa, suppose that R produces more data than T , but these additional data are not used by the rest of S . So, even if (the encodings of) R and T are not bisimilar, replacing R with T does not modify the external behaviour of the overall system, so these two services should still be considered interchangeable.

In order to get a general condition for replaceability, take Y as the subset of D_R containing those data neither in D_P nor used by the rest of the service S : formally, “being Y not used by S ” amounts to say that there exists an OCPR context $C[-]$ such that $\nu_Y[\|S\|_{D_S}] = C[\nu_Y[\|R\|_{D_R}]]$. Let us assume the existence of a subset Z of data of T such that $D_T \setminus Z = D_R \setminus Y$. Thus, we say that the replacement is *sound* (with respect to public specification P) if

$$\nu_Y[\|R\|_{D_R}] \approx_S \nu_Z[\|T\|_{D_T}]$$

The above condition amounts to say that the external behaviour of S does not change. Indeed, for $X = D_S \setminus D_P$ we have $\|P\|_{D_P} \approx_S \nu_X[\|S\|_{D_S}]$ (since P is the public specification of S), and requiring that $Y \subseteq X$ is not used in S implies

$$\|P\|_{D_P} \approx_S \nu_X[\|S\|_{D_S}] = \nu_{(X \setminus Y)}[C[\nu_Y[\|R\|_{D_R}]]] \approx_S \nu_{(X \setminus Y)}C[\nu_Z[\|T\|_{D_T}]]$$

So, the replacement is indeed sound. Finally, note that we may safely assume that the data in Z do not occur in S , possibly after some renaming, so that $Z \cap (D_S \setminus Y) = \emptyset$. By Proposition 2 we then obtain

$$\nu_{(X \setminus Y)}C[\nu_Z[\|T\|_{D_T}]] \approx_S \nu_{((X \setminus Y) \uplus Z)}C_Z[\|T\|_{D_T}]$$

which corresponds to the encoding of the process description S' , obtained after replacing R in S with T , and closing with respect to the data $(X \setminus Y) \uplus Z$.

5 Case Study (Continued)

In the last section we sketched a general methodology for the use of the theoretical results given in Section 3 in addressing the issues of service publication and service replaceability. The aim of this section is to directly instantiate the methodology on our case study.

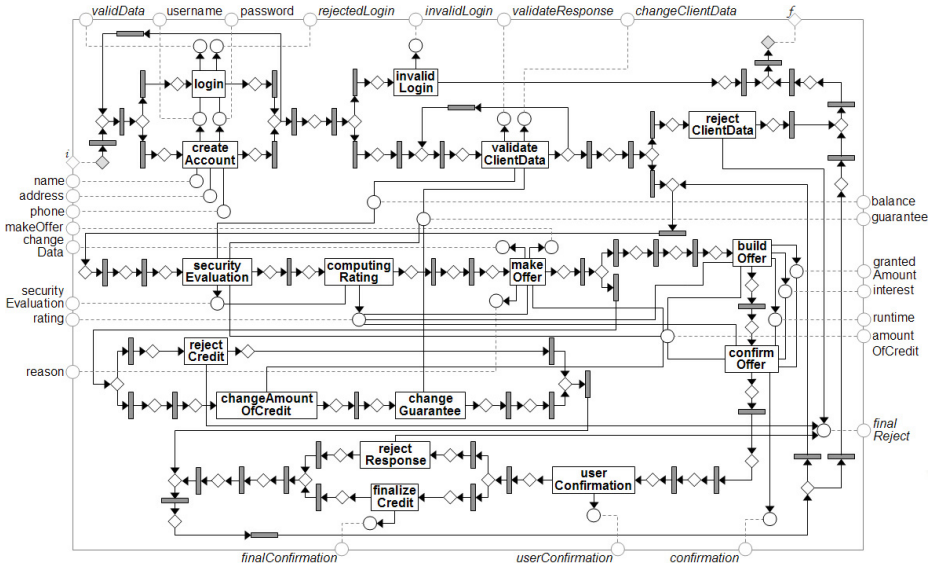


Fig. 7. *IMP*: OCPR net representation of the *CreditPortal* service

On service publication

Let us continue the first example of Section 2. As previously anticipated, the bank (i.e., the service provider) wants to verify whether the *interface behaviour description* of *CreditPortal* (Fig. 2) that it wants to publish properly advertises the full behaviour of the *CreditPortal* service (Fig. 1).

Firstly, we translate both the full process model and the interface behaviour description of *CreditPortal* into OCPR nets, according to the OWL-S encoding sketched in subsection 3.3. The resulting nets *IMP* and *SPEC* are illustrated in Figs. 7 and 8, respectively. As one may note, all the data places of the two nets are open. As a consequence, if we compare *IMP* with *SPEC* with respect to the behavioural equivalence of subsection 3.1, the two nets have different interfaces and they are hence externally distinguishable. As explained in Section 4, the correct way to proceed – before equating the nets – is to take a set X of data places that we do not want to be observed by the client. In the net *IMP* we would take $X = \{\text{securityEvaluation}, \text{rating}, \text{makeOffer}, \text{confirmation}\}$. After closing X , the two structurally different nets result to be externally indistinguishable, i.e., $\nu_X[\text{IMP}] \approx_S \text{SPEC}$. In other words, the simplified process model in Fig. 2 is a correct interface behaviour description for the *CreditPortal* service.

On service replaceability

In the second example of Section 2, the bank needs to verify whether the sub-service of *CreditPortal* which evaluates the customer reliability (the dotted area of Fig. 1) can be replaced by *RatingOne* (Fig. 3) or *RatingTwo* (Fig. 4) affecting neither the internal nor the external behaviour of *CreditPortal*.

Similarly to the previous example, we first translate the dotted area of *CreditPortal*, *RatingOne* and *RatingTwo* into the OCPR nets *SUB*, $\mathcal{R}_1$, $\mathcal{R}_2$ depicted

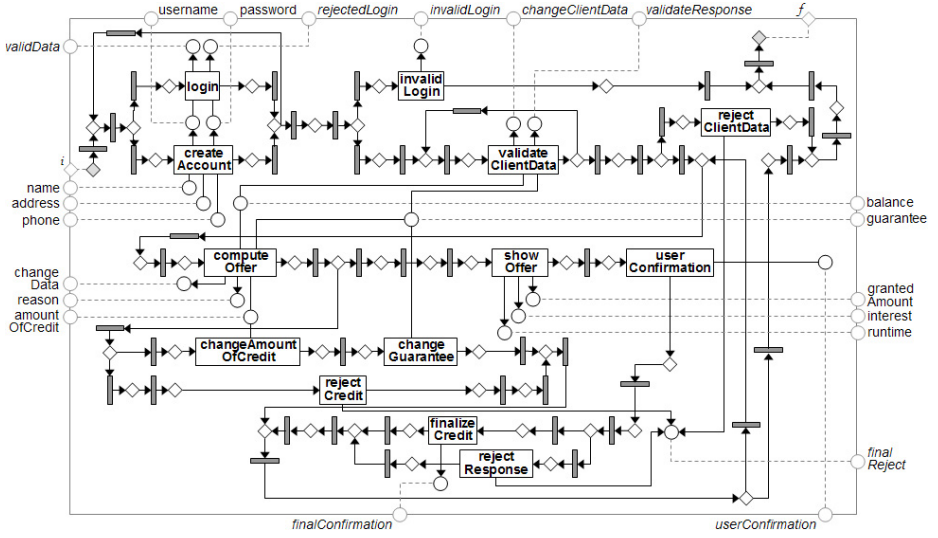


Fig. 8. *SPEC*: OCPN net representation of the CreditPortal public specification

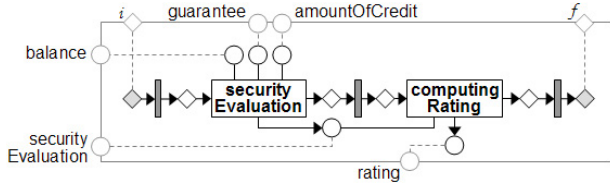
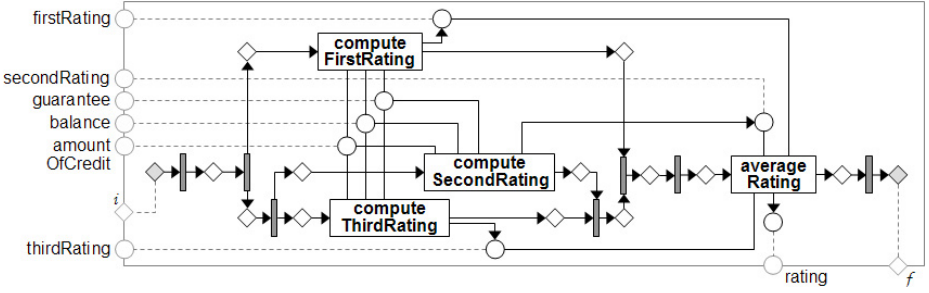
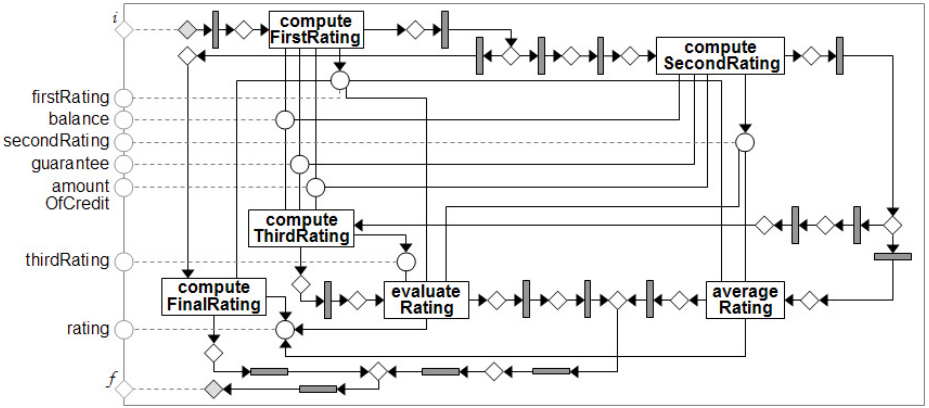


Fig. 9. *SUB*: the subservice of CreditPortal to be externalized

in Figs. 9, 10 and 11, respectively. Clearly, these three nets are not equivalent, since they expose different interfaces, and they are obviously externally distinguishable. This is not surprising: the bank describes which are the information it needs for each customer (namely, its rating), while the additional information (i.e., *firstRating*, *secondRating* and *thirdRating*) provided by the enterprises may be used by the bank in order to choose a service according to some criteria.

However both *RatingOne* and *RatingTwo* can safely replace the dotted area of *CreditPortal*. In particular, while such a (sub)service substitution affects the internal behaviour of *CreditPortal*, it does not alter its externally observable behaviour. Following the methodology sketched in Section 4, we take the set Y of data places of *SUB* that do not occur in the public specification and that are not used by the rest of the service. In our example, $Y = \{\text{securityEvaluation}\}$. Then we take Z as the set of data places occurring in $\mathcal{R}_1$ (or $\mathcal{R}_2$) and not in $\nu_Y[\text{SUB}]$. Thus, for both $\mathcal{R}_1$ and $\mathcal{R}_2$, $Z = \{\text{firstRating}, \text{secondRating}, \text{thirdRating}\}$.

At this point, we just need to check that $\nu_Y[\text{SUB}] \approx_S \nu_Z[\mathcal{R}_1] \approx_S \nu_Z[\mathcal{R}_2]$. Since the equivalences hold, both *RatingOne* and *RatingTwo* can be employed to replace the subservice of *CreditPortal* evaluating the customer reliability.

Fig. 10. $\mathcal{R}_1$: OCPN net representation of the RatingOne serviceFig. 11. $\mathcal{R}_2$: OCPN net representation of the RatingTwo service

6 Concluding Remarks

This paper outlines a methodology for addressing two pivotal issues in Service-Oriented Computing: *publication of correct service specifications* and *replaceability of (sub)services*. Given (the OWL-S process models of) a service S_1 , its (verified correct) public specification and a service S_2 , we want to check whether replacing a sub-component of S_1 with S_2 does not change the behaviour described in the specification. We thus translate the sub-component of S_1 and S_2 into OCPN nets and we check whether such nets are equivalent by closing those data places that do not occur in the public specification of S_1 . The key ingredients of the methodology are a *compositional* notion of saturated bisimilarity [3], its characterization *via* a *weak* and decidable bisimulation equivalence, a formal encoding from OWL-S to CPR nets (introduced here and implemented in [9]), and the definition of an hiding operator. The work is presented through an example scenario, inspired by the finance case study of the SENSORIA project.

In literature there are many approaches using Petri nets to model Web services. We discussed the issue in [3], where, in particular, we highlighted the

connection of OCPN nets to the workflow nets [10,11], and we pointed out the correspondence with the notion of simulation introduced by Martens in [12,13].

In the emerging world of Service-Oriented Computing – where applications are built by combining existing services – the issue of service replaceability gained a prominent role, and thus new approaches are often introduced. The discussion below briefly sums up some of the most recent proposals that we are aware of.

A logic-based approach for service replaceability has been recently presented in [14], where a context-specific definition of service equivalence is introduced. According to [14], given a μ -calculus formula ϕ describing some property, a service S , taking part in a specific context $C[-]$, can be replaced by a service T if ϕ holds also in $C[T]$. Intuitively, such a notion of context-specific replaceability relaxes the requirements imposed by a notion of service (bi)simulation like [3].

Another relaxed replaceability relation on services is induced by the definition of *interaction soundness* presented in [15]. Given an environment E , a service S in an orchestration $O[S]$ can be replaced by T if the interaction of $O[T]$ and E is *lazy sound*, that is, if the final node of the graph which represents the interaction of $O[T]$ and E can be reached from every initial node.

Although not presented in term of replaceability, the notion of *operating guidelines*, introduced in [16,17] and employed in [18] to formally analyze the interactional behaviour of BPEL processes, also implicitly induces a replaceability relation on services – yet not compositional. An operating guideline is an automaton that concisely represents all the partners that properly interact with a service. A service S interacting with C can be replaced with a service T if T belongs to the operating guidelines of C .

A theory for checking the compatibility of service contracts based on a CCS-like calculus is presented in [19,20]. Using a simple finite syntax (featuring the sequencing and external/internal choice constructors) to describe service contracts, they define a notion of preorder on processes (based on must testing) reflecting the ability of successfully interacting with clients. Such a notion induces a replaceability relation that, informally, allows one to replace a service S_1 with S_2 only if all clients compliant with S_1 are also compliant with S_2 . Such a notion of replaceability is uncomparable with ours, as the former emerges from a synchronous model while the latter emerges from an asynchronous model. It is also worth noting that, in particular, [20] shows the existence of the *principal dual contract* (reminiscent of operating guideline), i.e., the smallest (namely, the most general) service contract that satisfies the client request.

Other interesting notions of service replaceability were introduced also in [21, 22]. The approach in [21] models service behaviour as deterministic automata and defines substitutability with respect to three different notions of compatibility. In particular, *context dependent substitutability* states that given a service made of two sub-services S_1 and S_2 , S_1 can be replaced by S'_1 if S'_1 is compatible with S_2 , while *context independent substitutability* states that a service S can be replaced by a service S' if S' is compatible with all those services which are compatible with S (analogously to [12,20]). Our notion of substitutability resembles the notion of context independent substitutability (w.r.t. definition of

compatibility 1 of [21]) in an asynchronous and non-deterministic setting. The approach in [22] copes with timed business protocols and defines a notion of time-dependent compatibility/replaceability. Let P_1, P_2 be timed business protocols. Then, P_1 can replace P_2 w.r.t. a client protocol P_C if for each timed interaction trace of P_2 and P_C there is a corresponding timed interaction trace of P_1 and P_C . Yet, we do not consider time constraints in our notion of service replaceability.

Furthermore, our relying on the concept of bisimilarity allows us to benefit from the wealth of tools and algorithms developed so far. Indeed, we can check saturated bisimilarity by constructing a finite labelled transition system and then verifying weak bisimilarity there, exploiting e.g. the classical algorithm proposed in [23]. We are currently implementing such an solution. The expected worst-case time complexity can be roughly estimated in $O(S^2)$, where S denotes the number of the markings of an OCPR net. Indeed, given two OCPR nets, the time needed to construct their transition systems is $O(S)$, while the algorithm in [23] takes $O(S^2)$ for checking the weak bisimilarity. We intend however to develop a more efficient algorithm for checking saturated bisimilarity based on *normalized minimization* [24].

Finally, it is important to note that – for the sake of simplicity – we used a single range of names for identifying the parameters of the presented services, so that the mapping among parameters of different services is obvious. Yet, it is often the case that different services employ different parameter names. In the case of OWL-S descriptions, each functional parameter is annotated with a concept defined in a shared ontology. Hence, we can (semi-)automatically determine the mapping between parameters of separate services by employing suitable tools for crossing ontologies. Otherwise, in the case of WS-BPEL [25], for example, such a mapping has to be provided manually. In this perspective our approach can be easily extended to WS-BPEL services, exploiting, e.g., a translation from BPEL processes to workflow nets in [26].

References

1. Papazoglou, M., Georgakopoulos, D.: Service-Oriented Computing. Communications of the ACM 46(10), 24–28 (2003)
2. OWL-S Coalition: OWL-S: Semantic Markup for Web Service (2006), <http://www.ai.sri.com/daml/services/owl-s/1.2/overview/>
3. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web services. In: Arbab, F., Sarjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 240–256. Springer, Heidelberg (2007)
4. Corfini, S.: The CreditPortal service (2008), [http://www.di.unipi.it/corfini/owls/processmodels/\(economy\)_CreditPortal.owl.xml](http://www.di.unipi.it/corfini/owls/processmodels/(economy)_CreditPortal.owl.xml)
5. Reisig, W.: Petri Nets: An Introduction. EACTS Monographs on Theoretical Computer Science. Springer, Heidelberg (1985)
6. Brogi, A., Corfini, S.: Behaviour-aware discovery of Web service compositions. International Journal of Web Services Research 4(3), 1–25 (2007)
7. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Logic in Computer Science, pp. 69–80. IEEE Computer Society, Los Alamitos (2006)

8. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
9. Brogi, A., Corfini, S., Iardella, S.: From OWL-S descriptions to Petri nets. In: Nitto, E.D., Ripeanu, M. (eds.) ICSOC 2007 Workshops. LNCS, Springer, Heidelberg (to appear, 2008), <http://www.di.unipi.it/~corfini/paper/WESOA07.pdf>
10. Verbeek, H., van der Aalst, W.: Analyzing BPEL processes using Petri nets. In: Marinescu, D. (ed.) Applications of Petri Nets to Coordination, Workflow and Business Process Management, pp. 59–78. Florida International University, Miami (2005)
11. van der Aalst, W.: Pi calculus versus Petri nets: Let us eat “humble pie” rather than further inflate the “Pi hype”. BPTrends 3(5), 1–11 (2005)
12. Martens, A.: On compatibility of Web services. Petri Net Newsletter 65, 12–20 (2003)
13. Martens, A.: Analyzing Web service based business processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
14. Pathak, J., Basu, S., Honavar, V.: On context-specific substitutability of Web Services. In: Web Services, pp. 192–199. IEEE Computer Society, Los Alamitos (2007)
15. Puhlmann, F., Weske, M.: Interaction soundness for Service Orchestrations. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)
16. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
17. Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for finite-state services. In: Kleijn, Y., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
18. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
19. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. In: Ghelli, G. (ed.) Programming Language Technologies for XML, pp. 37–48 (2007)
20. Laneve, C., Padovani, L.: The must preorder revisited. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 212–225. Springer, Heidelberg (2007)
21. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 15–28. Springer, Heidelberg (2005)
22. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: Compatibility and replaceability analysis for timed web service protocols. In: Benzaken, V. (ed.) Bases de Données Avancées (2005)
23. Fernandez, J.C., Mounier, L., Jard, C., Jeron, T.: On-the-fly verification of finite transition systems. Formal Methods in System Design 1(2/3), 251–273 (1992)
24. Bonchi, F., Montanari, U.: Coalgebraic models for reactive systems. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 364–379. Springer, Heidelberg (2007)
25. BPEL Coalition: WS-BPEL 2.0 (2006), <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>
26. Ouyang, C., Verbeek, E., van der Aalst, W., Breutel, S., Dumas, M., ter Hofstede, A.: Formal semantics and analysis of control flow in WS-BPEL. Technical Report BPM-05-15, BPM Center (2005)

Modeling and Analysis of Security Protocols Using Role Based Specifications and Petri Nets

Roland Bouroulet¹, Raymond Devillers²,
Hanna Klaudel³, Elisabeth Pelz¹, and Franck Pommereau¹

¹ LACL, Université Paris Est, 61 av. du Gén. de Gaulle, F-94010 Créteil

² Département d'Informatique, CP212, Université Libre de Bruxelles

³ IBISC, Université d'Evry, bd François Mitterrand, F-91025 Evry

Abstract. In this paper, we introduce a framework composed of a syntax and its compositional Petri net semantics, for the specification and verification of properties (like authentication) of security protocols. The protocol agents (*e.g.*, an initiator, a responder, a server, a trusted third party, ...) are formalized as *roles*, each of them having a predefined behavior depending on their global and also local knowledge (including for instance public, private and shared keys), and may interact in a potentially hostile *environment*.

The main characteristics of our framework, is that it makes explicit, structured and formal, the usually implicit information necessary to analyse the protocol, for instance the public and private *context* of execution. The roles and the environment are expressed using SPL processes and compositionally translated into high-level Petri nets, while the context specifying the global and local knowledge of the participants in the protocol is used to generate the corresponding initial marking (with respect to the studied property). Finally, this representation is used to analyse the protocol properties, applying techniques of simulation and model-checking on Petri nets. The complete approach is illustrated on the case study of the Kao-Chow authentication protocol.

Keywords: security protocols, formal specification, process algebras, Petri nets.

1 Introduction

In the last years, security protocols have become more and more studied for their behaviours, causal dependencies and secrecy properties. Such protocols, where messages are asynchronously exchanged via a network, assume an underlying infrastructure, composed of (asymmetric) public, private and (symmetric) shared keys [30,31]. In order to specify more precisely security protocols, formal description languages have been proposed, like the *Spi-Calculus* in [3] or the *Security Protocol Language (SPL)* in [14]. The latter is an economical process language inspired from process algebras like the asynchronous π -Calculus [26]: each process is defined by a term of a specialized algebra which allows to represent sequences and parallel compositions of (more or less complex) input and output actions

which may contain messages. Thus, in contrast to other notations, sending and receiving messages can and have to be explicitly specified. However, when used to specify protocols with various agents, each of them playing a proper role with respect to the (global, but also local) knowledge it may have, an SPL presentation needs to be complemented in order to be verified. Usually, a number of implicit information is added, concerning for instance the global context of the protocol execution, the number and the identity of the participants, the knowledge about encryption keys each of them has, and so on.

In this paper, we provide a solution allowing to take into account all this important information in a structured way, making it explicit. We focus on the specification and verification of properties (like authentication) of security protocols. The active elements of these protocols may be formalized as *roles*, like for instance an initiator, a responder or a server, each of them having a predefined behavior taking place in a given *context*. They should interact in some potentially hostile *environment* in such a way that desired properties are preserved. Thus, in our approach, we define a framework in which a protocol is specified as a triple composed of a number of roles, an environment and a context. The roles and the environment are expressed using SPL processes, while the context specifies the knowledge of the participants in the protocol (including public, private and shared keys).

Next, this specification is translated into a class of composable high-level Petri nets. In particular, the roles and the environment are translated into *S-nets* [9,10]. All these nets are composed in parallel in order to get a single net. The context serves to define its initial marking (with respect to the studied property). Finally, this representation is used to analyse the protocol properties, applying techniques of simulation and model-checking [19].

With respect to previous translations into Petri nets [9,10], the present paper also introduces a way to use secret shared keys cryptography (while only public key cryptography was available before), and to take into account information about the past (like compromised old-session keys).

1.1 Case Study

The approach will be illustrated on the Kao-Chow (KC) authentication protocol [21,12], chosen as running example, involving participants exchanging messages using shared encryption keys. It implements three roles: those for an initiator, a responder and a server. As usual in a cryptographic context, all the participants make use of nonces, which are newly created values used by the participants to sign their messages.

The KC protocol between an initiator A , a responder B and a server S , allows A and B to mutually authenticate and to share a symmetric key K_{ab} used for secure communications after the agreement, *i.e.*, at the end of the protocol A and B are sure about the secrecy of their knowledge about K_{ab} . Informally and assuming that K_{as} and K_{bs} are symmetric keys initially known only by A and S , and respectively by B and S , the KC protocol may be described as follows:

1. $A \rightarrow S : A, B, m$

The initiator agent A sends to the server S its own agent name A , the name of the agent B with which it would like to communicate, together with a nonce m .

2. $S \rightarrow B : \{A, B, K_{ab}, m\}_{K_{as}}, \{A, B, K_{ab}, m\}_{K_{bs}}$

Then, S sends to B a message containing the received names A and B and the nonce m , together with the generated symmetric key K_{ab} , first encrypted with K_{as} , the symmetric key it shares with A , second with K_{bs} the one it shares with B . This second part, B should decrypt after reception.

3. $B \rightarrow A : \{A, B, K_{ab}, m\}_{K_{as}}, \{m\}_{K_{ab}}, n$

Then, B sends to A the first part of the received message (that it cannot decrypt because it is encrypted with K_{as}), together with the received nonce m encrypted with the shared (newly obtained) key K_{ab} , and its own nonce n . After reception, A decrypts the first part of the message, gets the key K_{ab} and checks it by decrypting its own nonce m .

4. $A \rightarrow B : \{n\}_{K_{ab}}$

Finally, A sends back to B the nonce n encrypted with K_{ab} .

This protocol suffers a similar kind of attack as the Denning Sacco freshness attack on the Needham Schroeder symmetric key protocol, when an older session symmetric key K_{ab} has been compromised.

1. $A \rightarrow S : A, B, m$

2. A and B previously used the symmetric key K_{ab} and we assume from now on that K_{ab} is compromised. So the following exchange has already occurred and can be replayed: $S \rightarrow B : \{A, B, m, K_{ab}\}_{K_{as}}, \{A, B, m, K_{ab}\}_{K_{bs}}$. An attacker I may now impersonate S and A to force B to reuse K_{ab} in a communication with what it believes to be A , while it is I .

3. $I(S) \rightarrow B : \{A, B, m, K_{ab}\}_{K_{as}}, \{A, B, m, K_{ab}\}_{K_{bs}}$

Agent I behaves as S and replays the same message to B .

4. $B \rightarrow I(A) : \{A, B, m, K_{ab}\}_{K_{as}}, \{m\}_{K_{ab}}, n'$

B executes the next step of the protocol and sends a message with a nonce n' to I , believing it is A .

5. $I(A) \rightarrow B : \{n'\}_{K_{ab}}$

I , knowing the symmetric key K_{ab} , encrypts n' and sends it back to B . Now B believes to have authenticated A and pursues a secret communication with I instead of A .

The corresponding role based specification of this protocol comprises three roles (for one initiator, one server and one responder), each of them being formalized as an SPL process together with the knowledge about keys that it knows. The environment is represented by a generic SPL process simulating a potentially aggressive attacker, and the global context is empty as the protocol does not assume any public knowledge. This specification is the basis of the compositional translation to high-level Petri nets for model-checking and simulation.

1.2 Outline

The paper is structured as follows: The next section presents syntactical aspects of our framework of role based specification, and their intuitive meaning. It includes a short presentation of the SPL syntax and its intuitive semantics, augmented by the explicit expression of keys, and the formal definition of a role based specification which uses SPL. The three elements of such a specification are described next, namely the initialized roles, the environment and the public context. This syntactical part is illustrated by providing a role based specification of the KC protocol, and its variants depending on the global context definition.

The next section introduces our Petri net (S-net) semantics of a role based specification and its compositional construction. In particular, the translation algorithm from SPL terms to S-nets for roles and for the environment is detailed as well as the one giving to the nets their initial marking. Dealing with symmetric keys, and possibly several of them between one pair of agents, needs a careful semantical treatment, which is explained and illustrated on small examples. Also, because of the modeling of a fully general environment, the translation potentially leads to infinite nets. Various ways allowing to cope with this problem are then presented and discussed.

Then, a section illustrates the use of the role based specification in order to automatically obtain a Petri net representation of KC protocol, and relates our verification experiences using existing model-checking tools.

Finally, we conclude by giving some indications about the directions of our future work.

2 Role Based Specification with SPL

First, we briefly recall the syntax and the intuitive semantics of SPL agents, inspired from [14].

2.1 Syntax and Intuitive Semantics of SPL

We assume countably infinite disjoint sets:

- of nonces (which are assumed to be numerical values) $N = \{n, n', n'', \dots\}$;
- of agents $G = \{A, B, \dots\}$;
- and of indexes for processes and keys $I = \{i, \dots\}$, containing in particular the natural numbers $\mathbb{N}$.

We distinguish also three disjoint sets of variables:

- $V_N = \{u, v, w, x, y, z, \dots\}$ for nonce variables;
- $V_G = \{X, Y, Z, \dots\}$ for agent variables;
- and $V_M = \{\Psi, \Psi', \Psi_1, \dots\}$ for message variables.

We use the vector notation $\vec{a}$ which abbreviates some possibly empty list of variables $a_1, \dots, a_l$ (in $V_N \cup V_G \cup V_M$), usually written in SPL as $\{a_1, \dots, a_l\}$. Also,

the notation $[\vec{V}/\vec{a}]$ represents the componentwise sort-preserving substitutions $a_i \mapsto V_i$, for $i \in \{1, \dots, l\}$.

The syntax of SPL is given by the following grammar:

$g ::= A, \dots \mid X \dots$	agent expressions
$e ::= n, \dots \mid u \dots$	nonce expressions
$k ::= \langle \text{Pub}, e, g \rangle \mid \langle \text{Priv}, e, g \rangle \mid \langle \text{Sym}, e, g_1, g_2 \rangle$	key expressions
$M ::= g \mid e \mid k \mid M_1, M_2 \mid \{M\}_k \mid \Psi$	messages
$p ::= \text{out new } \vec{u} \ M.p \mid \text{in pat } \vec{a} \ M.p \mid \parallel_{i \in I} p_i$	processes

where

- (a) $\langle \text{Pub}, e, g \rangle, \langle \text{Priv}, e, g \rangle, \langle \text{Sym}, e, g_1, g_2 \rangle$, where e is the actual value of the key, are used respectively for the public key of g , the private key of g , and the symmetric key shared by g_1 and g_2 ; we may also use indexes like $\langle \text{Pub}, e, g, i \rangle, \langle \text{Priv}, e, g, i \rangle, \langle \text{Sym}, e, g_1, g_2, i \rangle, i \in I$, if we need to manage several similar keys simultaneously. We shall denote by K the set of all possible private, public and symmetric keys without variables in the SPL format.
- (b) A message may be a name, a nonce, a key, a composition of two messages (M_1, M_2) , an encrypted message M using a key k , which is written $\{M\}_k$, or a message variable. By convention, messages are always enclosed in brackets $\{ \}$, thus allowing to distinguish between a nonce n and a message $\{n\}$ only containing it. Moreover, we denote by Msg the set of all messages without variables formed with respect to the SPL grammar, and $\mu(\text{Msg})$ the set of all multi-sets over it.
- (c) In the process $\text{out new } \vec{u} \ M.p$, the out-action prefixing p chooses fresh distinct nonces $\vec{n} = \{n_1, \dots, n_l\}$, binds them to variables $\vec{u} = \{u_1, \dots, u_l\}$ and sends the message $M[\vec{n}/\vec{u}]$ out to the network, then the process resumes as $p[\vec{n}/\vec{u}]$. It is assumed that all the variables occurring in M also occur in $\vec{u}$. The *new* construct, like in [14], ensures the freshness of values in $\vec{n}$. Notice that communications are considered as *asynchronous*, which means that output actions do not need to wait for inputs. By convention, we may simply omit the keyword *new* in an out-action if the list $\vec{u}$ of ‘new’ variables is empty.
- (d) In the process $\text{in pat } \vec{a} \ M.p$, the in-action prefixing p awaits for an input that matches the pattern M for some (sort-preserving) binding of the pattern variables $\vec{a}$, then the process resumes as p under this binding, *i.e.*, as $p[\vec{a}/\vec{a}]$, where $\vec{a}$ are the values in $N \cup G \cup \text{Msg}$ received in the input message. It is required that $\vec{a}$ is an ordering of all the pattern variables occurring in M . We may also omit the keyword *pat* in an in-action if the list $\vec{a}$ of ‘new’ variables is empty.
- (e) The process $\parallel_{i \in J} p_i$ is the parallel composition of all processes p_i with $i \in J \subseteq I$. The particular case where J is empty defines the process $\text{nil} = \parallel_{i \in \emptyset} p_i$; all processes should end with *nil* but, by convention, we usually omit it in examples. Replication of a process p , denoted $!p$, stands for an infinite composition $\parallel_{i \in \mathbb{N}} p$.

We shall denote by $\text{var}(p)$ the set of all the free variables occurring in the process p , *i.e.*, the variables used in p which are not bound by an in or out prefix. In order to simplify the presentation, we assume that all the sets $\overrightarrow{u}$ and $\overrightarrow{a}$ of variables occurring in the out and in prefixes of processes, as well as their sets of free variables, are pairwise disjoint.

2.2 Role Based Specification

We can now introduce our approach which mostly relies on notions like “role”, “environment”, and private and public “contexts”.

Given an SPL grammar as above, a *role* (*i.e.*, a named process) is a definition of the form $\delta(a_1, \dots, a_{m_\delta}) \stackrel{\text{df}}{=} p_\delta$, where p_δ is an SPL process, such that $\text{var}(p_\delta) \subseteq \{a_1, \dots, a_{m_\delta}\}$. For short, such a role is often identified with its name δ . Its context context_δ is defined as a pair (f_δ, l_δ) , where

- $f_\delta: \{a_1, \dots, a_{m_\delta}\} \rightarrow G \cup N \cup \text{Msg}$ is a sort-preserving mapping that associates an agent to each agent variable, a nonce (value) to each nonce variable and a message to each message variable;
- and $l_\delta \subseteq K$ is a subset of keys known by the role δ .

Intuitively, δ is the name of an agent, p_δ describes its behaviour, f_δ defines the instantiation of the process by fixing the values of the free variables and l_δ specifies the initial private knowledge of the agent (usually the agents initially know some keys; during the execution of the protocol they will be able to learn some more keys from the messages received through the network). We shall denote by $(\delta(f_\delta), l_\delta)$ an initialised role δ together with its private context.

Besides agents, the system will also encompass some spies and the network interconnecting all of them.

A *role based specification* is defined as a triple

$$(\text{Initialized\_Roles}, \text{Environment}, \text{Public\_Context}),$$

where

- $\text{Initialized\_Roles} \subset \{(\delta(f_\delta), l_\delta) \mid \delta \text{ is a role}\}$ is a (finite) non-empty set of initialised roles with their private contexts;
- $\text{Environment} \subset \{s \mid s \text{ is a role}\}$ defines an environment, *i.e.*, a set of prototype roles (spies) whose replication and composition will allow to form more complex attackers; note that spies do not have a context, since they have no free variable, nor private knowledge (in order to cooperate, they make public all they know);
- $\text{Public\_Context} \in \mu(\text{Msg})$ is the initial public context, *i.e.*, a (multi-)set of messages previously sent on the network and available to everyone (to spies as well as to agents) when the protocol starts; it comprises usually the names of the agents in presence, but also possibly compromised keys, etc.

Using a role based specification to express a security protocol hence consists in defining the roles corresponding to all the participants (initiator, responder,

trusted third party, ...) and the roles corresponding to the prototypes of the potential attackers (environment). Potential attackers may be described using *spy* processes. Such processes will have the possibility of composing eavesdropped messages, decomposing messages and using cryptography whenever the appropriate keys are available. Below we present the six basic SPL spy processes, which refer to the standard Dolev-Yao [17] model. They are inspired from [14]. By choosing various specifications for the attacker (*i.e.*, various combinations of spy processes) one can restrict or increase its power of aggressiveness.

The following six spy processes may be divided into three groups depending on the action they perform. Thus, the first group deals with messages composition/decomposition: *Spy*₁ reads two messages from the network and issues to the network the message composed of the two read messages, while *Spy*₂ decomposes a message read on the network and issues its parts.

$$\begin{aligned} Spy_1 &\stackrel{\text{df}}{=} in\ pat\ \{\Psi_1\}\{\Psi_1\}. in\ pat\{\Psi_2\}\{\Psi_2\}. out\ \{\Psi_1, \Psi_2\} \\ Spy_2 &\stackrel{\text{df}}{=} in\ pat\ \{\Phi_1, \Phi_2\}\{\Phi_1, \Phi_2\}. out\ \{\Phi_1\}. out\ \{\Phi_2\} \end{aligned}$$

The next group deals with encryption: *Spy*₃ encrypts a message with the public key of someone, and issues it to the network, while *Spy*₄ encrypts a message with a symmetric key, and issues it.

$$\begin{aligned} Spy_3 &\stackrel{\text{df}}{=} in\ pat\ \{u, g\}\{\langle Pub, u, g \rangle\}. in\ pat\{\Psi\}\{\Psi\}. out\ \{\Psi\}_{\langle Pub, u, g \rangle} \\ Spy_4 &\stackrel{\text{df}}{=} in\ pat\ \{u, g_1, g_2\}\{\langle Sym, u, g_1, g_2 \rangle\}. in\ pat\{\Psi'\}\{\Psi'\}. out\ \{\Psi'\}_{\langle Sym, u, g_1, g_2 \rangle} \end{aligned}$$

The last group deals with decryption thanks to some obtained keys: *Spy*₅ decrypts a message with the private key of someone if it is available, and issues it to the network, while *Spy*₆ decrypts a message with a symmetric key if it is available, and issues it.

$$\begin{aligned} Spy_5 &\stackrel{\text{df}}{=} in\ pat\ \{u, g\}\{\langle Priv, u, g \rangle\}. in\ pat\{\Phi\}\{\Phi\}_{\langle Pub, u, g \rangle}. out\ \{\Phi\} \\ Spy_6 &\stackrel{\text{df}}{=} in\ pat\ \{u, g_1, g_2\}\{\langle Sym, u, g_1, g_2 \rangle\}. in\ pat\{\Phi'\}\{\Phi'\}_{\langle Sym, u, g_1, g_2 \rangle}. out\ \{\Phi'\} \end{aligned}$$

In case we need to manage more than one key of a kind per agent, the spy definitions take this information into account, for example,

$$Spy_3 \stackrel{\text{df}}{=} in\ pat\ \{u, g, i\}\{\langle Pub, u, g, i \rangle\}. in\ pat\{\Psi\}\{\Psi\}. out\ \{\Psi\}_{\langle Pub, u, g, i \rangle}$$

In this model, a general potentially very aggressive attacker is modeled by an infinite set of spy processes running in parallel.

2.3 A Role Based Specification of the KC Protocol

The roles in our modeling of the KC protocol comprise the following SPL process definitions, where $X, X', X'', \dots$, are agent variables in V_G , $u, u', u'', \dots$, are nonce variables in V_N , and $\Psi \in V_M$ is a message variable:

$$\begin{aligned}
Init(X, Y, Z, u) &\stackrel{\text{df}}{=} out\ new\ \{x\}\ \{X, Y, x\}. \\
in\ pat\ \{y, z\} &\ \{\{X, Y, \langle Sym, z, X, Y \rangle, x\}_{\langle Sym, u, X, Z \rangle}, \{x\}_{\langle Sym, z, X, Y \rangle}, y\}. \\
out\ \{y\} &\ \{y\}_{\langle Sym, z, X, Y \rangle}\} \\
\\
Serv(Z', u', v') &\stackrel{\text{df}}{=} in\ pat\ \{X', Y', x'\}\ \{X', Y', x'\}. \\
out\ new\ \{w'\} &\ \{\{X', Y', \langle Sym, w', X', Y' \rangle, x'\}_{\langle Sym, u', X', Z' \rangle}, \\
&\ \{X', Y', \langle Sym, w', X', Y' \rangle, x'\}_{\langle Sym, v', Y', Z' \rangle}\} \\
\\
Resp(Y'', Z'', u'') &\stackrel{\text{df}}{=} \\
in\ pat\ \{X'', y'', w'', \Psi\} &\ \{\Psi, \{X'', Y'', \langle Sym, w'', X'', Y'' \rangle, y''\}_{\langle Sym, u'', Y'', Z'' \rangle}. \\
out\ new\ \{x''\} &\ \{\Psi, \{y''\}_{\langle Sym, w'', X'', Y'' \rangle}, x''\}. \\
in\ \{x''\} &\ \{x''\}_{\langle Sym, w'', X'', Y'' \rangle}
\end{aligned}$$

A complete role based specification of the KC protocol comprising one initiator A , one server S and one responder B is given by

$$\begin{aligned}
&(\ \{ (Init(A, B, S, n_1), \{\langle Sym, n_1, A, S \rangle\}), \\
&\quad (Serv(S, n_1, n_2), \{\langle Sym, n_1, A, S \rangle, \langle Sym, n_2, B, S \rangle\}), \\
&\quad (Resp(B, S, n_2), \{\langle Sym, n_2, B, S \rangle\})\ \}, \\
&\quad \{ Spy_1, Spy_2, Spy_4, Spy_6\ \}, \\
&\quad \mathcal{M} \cup \{\{A\}, \{B\}, \{S\}\}\ \},
\end{aligned}$$

where the roles $Init(X, Y, Z, u)$, $Serv(Z', u', v')$, $Resp(Y'', Z'', u'')$ and Spy_i are defined as above, and $\mathcal{M}$ is a set of residual messages. Here we use only the spy processes that make sense in the case of the KC protocol (for instance, Spy_3 deals with public key cryptography that is not involved in KC).

2.4 Different Versions of the KC Protocol

We will consider in the following two versions of the KC protocol and its environment. The first one will be used to check the intrinsic resistance of the protocol, and will start from an empty initial knowledge: $\mathcal{M} = \emptyset$. But since the essence of the KC protocol is to manage short term session keys, it could happen that a key becomes compromised after some time. This may happen, for example, due to more or less clever brute force attacks. So, we will consider a version where $\mathcal{M}$ contains initially such a compromised key. Another KC variant would be a system with various initiators and responders.

3 Petri Net Translation of a Role Based Specification

3.1 The General Picture

Given a role based specification, its Petri net semantics may be obtained compositionally using an algebra of high-level Petri nets, inspired from the S-nets

introduced in [9] and its optimised version defined in [10]. Basic (unmarked) nets in this algebra allow to directly represent the SPL prefixes and are one-transition nets, like those in Figures 1 and 2, where the places represent composition interfaces, which are either *control-flow* (entry, exit or internal) or *buffer* ones. The labeling of the last ones (that are devoted to asynchronous communications) may be either:

- SPL variable names (each such place storing the current value of the variable);
- κ_δ (storing information about encryption/decryption keys privately known by a role δ);
- or Ω (storing all the messages transferred through the public network).

These nets may be composed with various composition operations:

- sequentially ($N_1; N_2$, which means that the exit places of the first net and the entry places of the second net are combined);
- in parallel ($N_1 \parallel N_2$, which means that both nets are put side by side);
- or replicated ($!N = \parallel_{i \in \mathbb{N}} N$).

All these operations include the merging of buffer places having the same label in order to allow asynchronous communications.

Thus, if $(\delta(f_\delta), l_\delta)$ is an initialized role in a role based specification, the process expression that describes the behavior of δ allows to compositionally construct a corresponding unmarked S-net. Its initial marking is obtained by putting a (black) token into its entry places while the tokens in the buffer places are put accordingly to the context (f_δ, l_δ) .

If *Spy* is a prototype role defining a kind of activity of the environment, we proceed similarly but implicitly assume that the parallel composition of all such roles may be replicated. Its finite structure representation is then obtained as an unmarked S-net, as defined in [10]. Its initial marking is defined by putting one token in each entry place and no token elsewhere. The complete Petri net representation of a role based specification is then a parallel composition of all the corresponding S-nets. Its initial marking includes the markings mentioned above and the tokens (messages) coming from the public context in the place Ω .

3.2 Translation Scheme

Let us assume that

$$(\{(\delta_1(f_1), l_1), \dots, (\delta_p(f_p), l_p)\}, \{s_1, \dots, s_r\}, \text{Public\_Context})$$

is a role based specification, where each role δ_i is defined as

$$\delta_i(a_1, \dots, a_{m_{\delta_i}}) \stackrel{\text{df}}{=} p_{\delta_i}.$$

Its translation into Petri nets is defined by the following phases, where the semantic function **Snet** will be detailed later on:

- *Phase 1.* Each role definition δ_i is translated first into an S-net $\text{Snet}(\delta_i)$, then we construct a net R as the parallel composition $\text{Snet}(\delta_1) \parallel \dots \parallel \text{Snet}(\delta_m)$, which merges in particular all the buffer places Ω and all the key places κ_{δ_i} for the same δ_i ;
- *Phase 2.* Each role definition s_j (spy) is translated first into an optimized transformed S-net $\text{Snet}(s_j)$. Then, we construct a net SPY as the parallel composition $\text{Snet}(s_1) \parallel \dots \parallel \text{Snet}(s_r)$, which encodes every possible execution of the attackers. Note that this translation is similar for some class of specifications and may be performed only once;
- *Phase 3.* The nets R and SPY are composed in parallel $R \parallel \text{SPY}$ and marked as follows:
 - the place Ω receives all the tokens (messages) from *Public_Context*;
 - each entry place receives a black token;
 - for each initialized role δ_i :
 - * each key place κ_{δ_i} receives all the tokens from l_i ;
 - * each place a_j , for $0 \leq j \leq m_{\delta_i}$ receives the token $f_i(a_j)$.

3.3 Petri Net Semantics of Roles

Each role definition $\delta(\dots) \stackrel{\text{df}}{=} p_\delta$ may be translated compositionally into a corresponding S-net, as defined in [9], providing its high-level Petri net representation. S-nets, like other high-level Petri net models, carry the usual annotations on places (types, *i.e.*, sets of allowed tokens), arcs (multisets of net variables) and transitions (guards, *i.e.*, Boolean expressions that play the role of occurrence conditions). S-nets have also the feature of read-arcs, which are a Petri net specific device allowing transitions to check concurrently for the presence of tokens stored in their adjacent places [11]. Like other box-like models [6,7], S-nets are also provided with a set of operations giving them an algebraic structure. In the version considered here, this is achieved through an additional labeling of places which may have a status (entry, exit, internal or buffer) used for net compositions.

The marking of an S-net associates to each place a multiset of values (tokens) from the type of the place and the transition rule is the same as for other high-level nets; namely, a transition t can be executed if the inscriptions of its input arcs can be mapped to values which are present in the input places of t and if the guard of t , $\gamma(t)$, is true under this mapping. The execution of t transforms the marking by removing values (accordingly to the mapping of arc inscriptions) from the input places of t and by depositing values into its output places. Read-arcs are only used to check the presence of tokens in the adjacent places but do not modify the marking. They are represented in figures as undirected arcs. Notice that given a transition t and a marking μ , there may be several possible executions (firings), corresponding to different mappings of variables in the inscriptions around t to values (called *bindings* and denoted σ).

More precisely, we distinguish the following sets of tokens as types for the places in S-nets.

- $\{\bullet\}$ for the entry, internal and exit places;
- **Msg** for the buffer place labeled Ω intended to gather all the messages present in the network, as well as for all buffer places labeled with SPL message variables (from V_M), even if the latter are intended to carry at most one message. For implementation or efficiency reasons, the set **Msg** may be restricted, for instance to a given maximal message length;
- $N \cup G \cup K \cup \mathbf{Msg}$ for all the other buffer places.

The S-net transitions always have at least one input and at least one output control-flow place; they may be of two sorts, corresponding to the two basic SPL actions ‘in’ and ‘out’. Arcs are inscribed by sets of net variables (denoted $a, b, c, d, \dots$). In order to avoid too complex (hence difficult to read) transition guards in some figures, we use a shorthand allowing to put complex terms as arc inscriptions (which has no consequences on the behavior of the net): we may thus replace an arc annotation c and a guard $c = f(a, b, d)$ by the arc annotation $f(a, b, d)$ and the true (or empty) guard.

By analogy with the two basic SPL actions, there are two basic S-nets. Each of them has a single transition, inscribed **IN** or **OUT**, one entry and one exit place of type $\{\bullet\}$, one buffer place labeled Ω of type **Msg**, a unique key place κ_δ where δ is the role in which the modeled action occurs, and some other buffer places, as shown in Figures 1 and 2.

For instance, the SPL output action

$$\text{out new } \{x''\}\{\Psi, \{y''\}\}_{\langle \text{Sym}, w'', X'', Y'' \rangle}, x''\}$$

in the role $\text{Resp}(Y'', Z'', u'')$ gives rise to the S-net represented in Figure 1. This net may be executed if there is a token $\bullet$ in the entry place (denoted by an incoming $\Rightarrow$), and if the buffer places labeled $w'', y'', X'', Y'', Z'', \Psi$ and

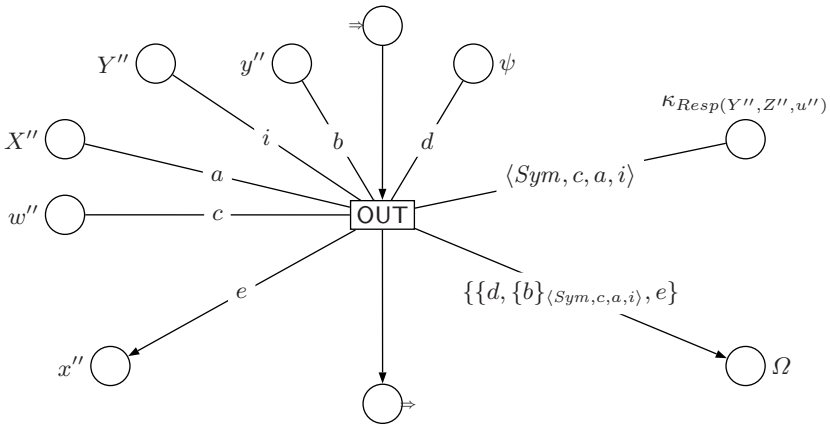


Fig. 1. The S-net corresponding to $\text{out new } \{x''\}\{\Psi, \{y''\}\}_{\langle \text{Sym}, w'', X'', Y'' \rangle}, x''\}$ in the role $\text{Resp}(Y'', Z'', u'')$

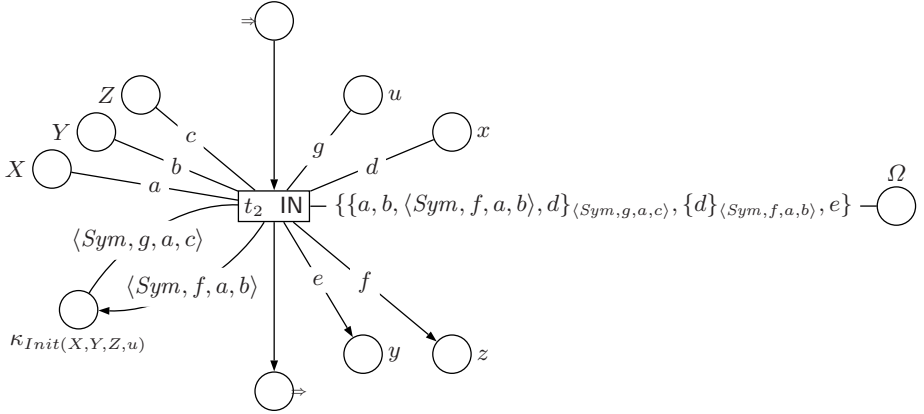


Fig. 2. The S-net for the ‘in’ action in the role $Init(X, Y, Z, u)$

$\kappa_{Resp}(Y'', Z'', u'')$, are marked (*i.e.*, contain values). At the firing of the transition under a binding σ , a nonce¹ is generated and put in place x'' ; the tokens existing in places w'' , y'' , X'' , Y'' , Z'' , Ψ and $\kappa_{Resp}(Y'', Z'', u'')$ are read (but not removed); the token $\bullet$ is transferred from the entry place to the exit place (denoted by an outgoing $\Rightarrow$); and the whole message $\{\sigma(d), \{\sigma(b)\}_{\langle Sym, \sigma(c), \sigma(a), \sigma(i) \rangle}, \sigma(e)\}$ is put in the message buffer Ω .

The elementary SPL input action

$$in\ pat\ \{y, z\} \{ \{X, Y, \langle Sym, z, X, Y \rangle, x\}_{\langle Sym, u, X, Z \rangle}, \{x\}_{\langle Sym, z, X, Y \rangle}, y \}$$

of the role $Init(X, Y, Z; u)$ is obtained analogously and represented in Figure 2. The message $\{ \{X, Y, \langle Sym, z, X, Y \rangle, x\}_{\langle Sym, u, X, Z \rangle}, \{x\}_{\langle Sym, z, X, Y \rangle}, y \}$ has local nonce variables y , z and u , global nonce variables x , X , Y , Z and no message variable. At the firing of the transition, a message from Ω with the adequate pattern is read, such that the existing values of x , X , Y and Z in the corresponding places are checked (through the read-arcs) to be equal to the corresponding elements of the message, and the value of the variables y and z are decoded from the message and put in the corresponding places.

The S-net representing a role δ is defined compositionally using S-net composition operations and is denoted $Snet(\delta)$.

We can observe, that the place Ω is part of each basic net and, through merging, thus becomes global to each composed net, like in Figures 5, 6 and 7.

3.4 S-Net Semantics of the Environment

The most general environment of a protocol is the most aggressive attacker, thus an unbounded number of all sorts of spies. Its role SPY can be described as a

¹ We assume that σ is such that $\sigma(e)$ is fresh. This can be implemented by using a place initially marked by a set of nonce values that are consumed when a fresh one has to be generated.

replication of all involved spies in parallel (for KC we have $J = \{1, 2, 4, 6\}$, 3 and 4 being irrelevant in this context).

$$\text{SPY} = !(\parallel_{i \in J} \text{Spy}_i) \text{ or equivalently } \text{SPY} = \parallel_{i \in J} ! \text{Spy}_i.$$

The associated net $\text{Snet}(\text{SPY})$ is that obtained by the infinite parallel composition of the S-nets obtained via the standard translation of the various Spy_i 's.

For instance, Figure 3 shows the standard translation of our first spy.

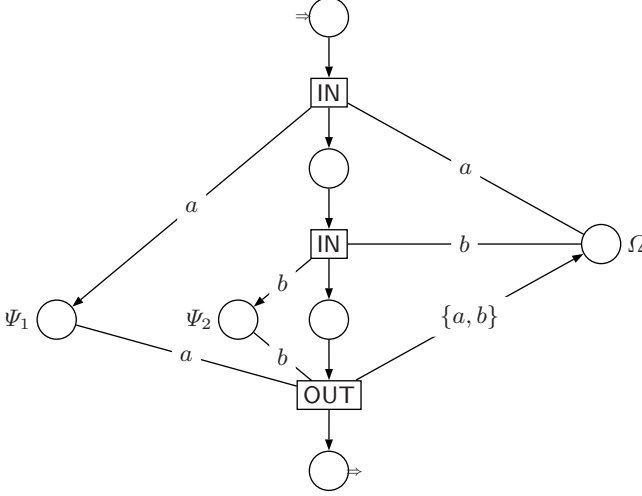


Fig. 3. One copy of the S-net of Spy_1

This representation and translation of the environment presents some difficulties with respect to verification: the process SPY representing all potential attackers is infinite, as well as $\text{Snet}(\text{SPY})$. A way to cope with this problem is to propose net transformations which do not essentially modify the net behaviour but lead to finite net and marking graph. This is possible, as shown in [10], by proposing solutions to the following three problems, given by increasing difficulty:

1. The composed messages do not have a bounded length.
2. The net is infinite (even for a bounded number of agents) because of replication in SPY .
3. The multiplicity of tokens in place Ω is not bounded.

The first and third ones induce infinite paths in the marking graph, the second one infinite edge degrees. We just like to quote here the adopted solutions which ensure finiteness.

1. The length of composed messages will be bounded by a (usually very small) constant, which corresponds to the maximum length of valid messages in the considered protocol; hence, the type Msg of buffer places will be a bit restricted, but the messages not representable are useless for the protocol.

2. Replication will be replaced by iteration, in the form of a loop. Thus we have only to consider $|I|$ transformed Spy_i -nets which are cyclic, and where at each cycle the buffer places which should initially be empty are emptied by a “flush” transition. The resulting behaviour is not exactly the same as with a true replication, since no two copies of a same spy may be simultaneously in the middle of their work; but this has no impact on the messages they can produce in Ω .
3. The place Ω will contain a multi-set with a very small bound of multiplicities. This can be achieved by adding a place am_i (for access-memory) to each Spy_i net, containing a local copy of Ω : its type is organized as a list without repetition, ensuring (via the guard of the upper transition) that no multiple copies are put in the global place Ω by this Spy . (Identical copies may be still be produced by different spies, but there is a small number of them.) Additional places (see ss_1 in Figure 4) allow to force the next transitions to use the same values as the first one when necessary.

This way, we obtain an optimized transformed spy-net for each Spy_i , called $\text{Snet}(Spy_i)$, as illustrated (for $i = 1$) in Figure 4. Via an adequate equivalence relation on these nets, it has been shown in [10], that these transformations preserve the (interesting part of the) original behaviour: $\text{Snet}(Spy_i) \equiv ! Spy_i$.

By taking $\text{Snet}(\text{SPY}) = \parallel_{i \in I} \text{Snet}(Spy_i)$ we have a finite net representation of the complete SPY role.

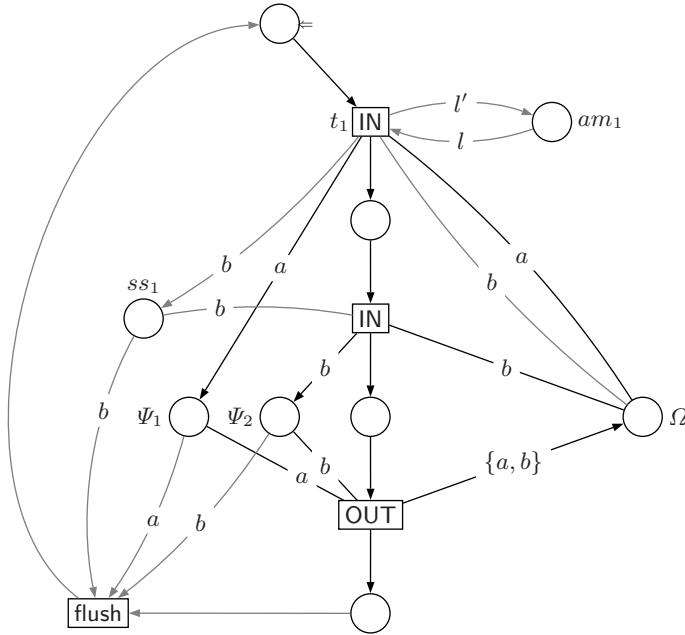


Fig. 4. The net $\text{Snet}(Spy_1)$, where the guard of t_1 is $((a.b) \notin l) \wedge l' = l.(a, b)$

4 Petri Net Model of the KC Protocol and Its Verification

4.1 S-Net Semantics of KC Agents

There are three agents involved in the KC protocol: the initiator, the server and the responder. The first one is represented by the role $Init(X, Y, Z, u)$, where X denotes the name of the agent itself, Y the name of the responder and Z the name of the server involved. The variable u denotes the value of the symmetrical key that the initiator shares with the server. So we get for the initiator the S-net depicted in Figure 5.

The second one is the server, represented by the role $Serv(Z', u', v')$, whose free variables are Z' , u' , v' , denoting respectively the name of the server itself, and two key values that the server will distribute on receiving the first message from the initiator. The corresponding S-net is depicted in Figure 6.

Finally, the responder corresponds to the role $Resp(Y'', Z'', u'')$, where Y'' represents the name of the responder itself, Z'' to the server and u'' is the value

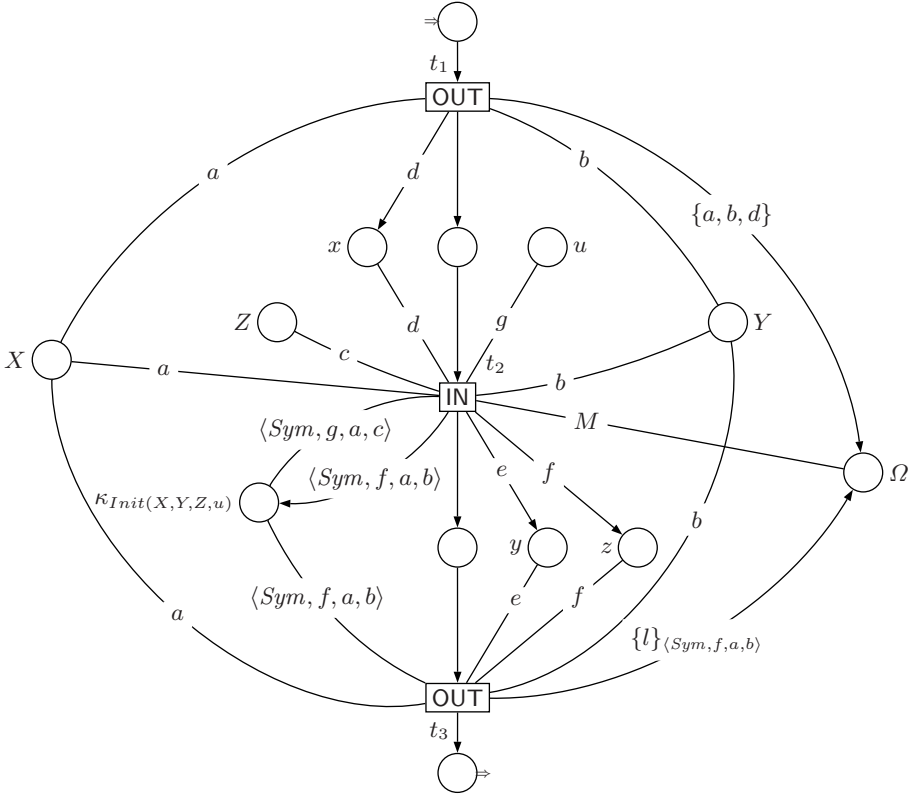


Fig. 5. The S-net for $Init(X, Y, Z, u)$, where the guard of t_2 is the equality $M = \{\{a, b, \langle Sym, f, a, b \rangle, d\}_{\langle Sym, g, a, c \rangle}, \{d\}_{\langle Sym, f, a, b \rangle}, e\}$

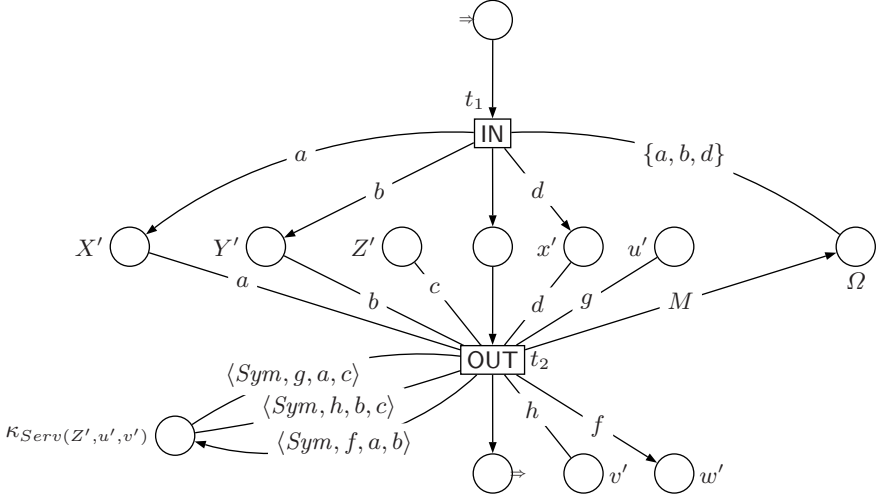


Fig. 6. The S-net for $Serv(Z', u', v')$, where the guard of t_2 is $M = \{\{a, b, \langle Sym, f, a, b \rangle, d\}_{\langle Sym, g, a, c \rangle}, \{a, b, \langle Sym, f, a, b \rangle, d\}_{\langle Sym, h, b, c \rangle}\}$

of the key that is shared between the Z'' and Y'' . The corresponding S-net is shown in Figure 7.

The S-net of the complete protocol is made of the agents and the environment $Snet(SPY)$, as explained above.

4.2 Automated Verification of KC

The next step of our approach is the automated verification of the authentication property of KC using model-checking techniques. We used for that the Helena analyzer [19], which is a general purpose high-level Petri net model-checker.

Starting from the role based specification, as described above, we implemented in Helena the S-nets of the roles and of the environment.

We considered first the intrinsic resistance of the KC protocol, *i.e.*, the variant where the public context $\mathcal{M}$ is empty. In order to test the resistance of the protocol in such an environment, we used a complete protocol with two agents A , B and a server S , with the environment formed of Spy_1 , Spy_2 , Spy_4 and Spy_6 .

In the KC protocol, B authenticates A when it receives the last message $\{n\}_{K_{ab}}$ from A , while A authenticates B on receiving the message

$$\{A, B, K_{ab}, m\}_{K_{as}}, \{m\}_{K_{ab}}, n.$$

That implies B authenticates after A does. In the net vocabulary, agents A and B mutually authenticate when the exit places of $Snet(Init(A, B, S, n_1))$ and $Snet(Resp(B, S, n_2))$ are marked. Therefore, a first property to check is that it is not possible to reach a marking where the exit place of $Snet(Resp(B, S, n_2))$

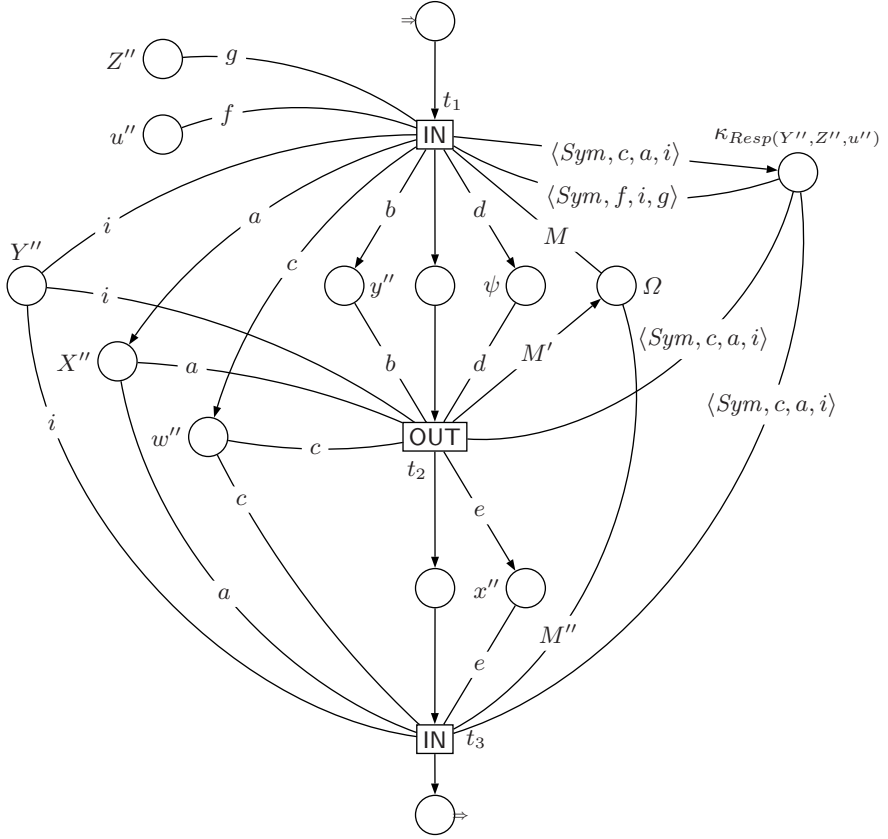


Fig. 7. The S-net for $\text{Resp}(Y'', Z'', u'')$, where the guard of t_1 is $M = \{d, \{a, i, \langle \text{Sym}, c, a, i \rangle, b \}_{\langle \text{Sym}, f, i, g \rangle}\}$, the guard of t_2 is $M' = \{\{d, \{b \}_{\langle \text{Sym}, c, a, i \rangle}, e\}\}$, and the guard of t_3 is $M'' = \{e\}_{\langle \text{Sym}, c, a, i \rangle}$

is marked while the exit place of $\text{Snet}(\text{Init}(A, B, S, n_1))$ is not. This verification was successfully performed: no such violating marking is reached when the initial knowledge is empty.

The second analysis was to check if the protocol still resists when the public context is not initially empty. It means that agents A and B have already used the KC protocol. So, some of the messages exchanged during previous sessions are part of the public context and we can assume also that for some reason a previous key session is compromised and was made public by some other (brute force, for instance) attacker. Therefore, it means in terms of the net semantics, that the place Ω contains the messages $\{\langle \text{Sym}, n_3, B, S \rangle\}$ and

$$\{\{A, B, \langle \text{Sym}, n_3, A, B \rangle, a\}_{\langle \text{Sym}, n_1, A, Z \rangle}, \{A, B, \langle \text{Sym}, n_3, A, B \rangle, a\}_{\langle \text{Sym}, n_2, B, Z \rangle}\}$$

together with the messages containing the names of the agents involved in the protocol $\{A\}, \{B\}, \{S\}$, as usual.

In this case, violating markings are the same as previously: the exit place of $\text{Snet}(\text{Resp}(B, S, n_2))$ is marked and the exit place of $\text{Snet}(\text{Init}(A, B, S, n_1))$ is not. In that case we found that the violating marking was reached and therefore this implies that the authentication property is violated. This corresponds to the fact that the spies succeeded in convincing B that a successful new session has been started with A , while this is not the case.

It took 29s using the Helena model-checker to compile and analyse the entire system in both cases on an Intel® Core™ Duo 2.33GHz.

5 Conclusion and Future Work

Security protocols are a very delicate field. Since they imply concurrent agents communicating through a generally public and insecure medium (allowing attackers to add, suppress and forge messages), they are prone to numerous forms of weaknesses, difficult to track and discover. And indeed, it is not uncommon that a protocol seems (sometimes “trivially”) correct and is used for years before a flaw is discovered. A famous example of this kind is for instance the Needham-Schroeder protocol [28], broken 17 years after its introduction [23,24].

Hence it appears necessary to use more formal and systematic ways to assess such protocols [25], like for instance [2,22,13,8]. Several approaches, like theorem provers or model-checking, were applied and sometimes offered in integrated environments like CAPSL [15] or AVISPA [1]. Also, type-checking [20,2] has been recently used, which like model-checking has the advantage to be completely automatic, but since security violations are defined in terms of type inconsistencies, the security property to be proved has to be considered when the specification is being written. Among various model-checking approaches, one may quote for instance Murφ [16,27] (based on condition/action rules and state space exploration), Casper [18] (generating CSP specification for the FDR model-checker) or Athena [32,33] (based on the strand space model, specialised logic and model-checker). They are mostly not process but message transfer oriented, often consider intruder implicitly and do not always explicitly express local and global knowledge.

Since Petri nets were especially introduced to cope with concurrency problems (the interleaving semantics is often enough to do so, but true concurrent semantics are also available) and to model accesses to common resources (through places of low-level or high-level nets), and since effective tools are now available to analyse and model-check them, we felt appealing to use this approach, like [29,5].

We thus introduced, and illustrated on a concrete case study (the Kao-Chow authentication protocol), a role based specification framework devoted to the specification and verification of properties of security protocols. It is composed of three uniformly defined parts describing the behaviour of the roles, the behaviour of the environment (representing a potentially aggressive attacker), and the global and local knowledges (about agent names, messages, or public, private and shared keys) the roles can have.

Unlike usual security protocol notations do, that is similar to what we used in the introduction, the advantage of a role based specification is that it is fully explicit and formal. The roles and the environment are expressed using SPL processes provided with their private contexts, and a definition of a global context. These SPL descriptions are then compositionally translated into high-level Petri nets, while the context (depending on the studied property) is used to generate the corresponding initial marking. An immediate advantage of the method is that the obtained Petri net model can be analysed using standard model-checking or simulation tools.

Compared to our previous approaches [9,10], the present paper introduce novel features for the treatment of symmetric keys as well as asymmetric ones, together with a uniform way to analyse various forms of attack contexts.

We already started to use this framework to analyse other kinds of protocols, mixing both symmetric and asymmetric keys, and allowing other kinds of attack contexts. This should be the subject of forthcoming papers.

References

1. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
2. Abadi, M., Blanchet, B.: Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM* 52(1) (2005)
3. Abadi, M., Gordon, A.: A calculus for cryptographic protocols. In: *The Spi calculus*. ACM Conference on Computers and Communication Security. ACM Press, New York (1997)
4. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology* 15(2) (2002)
5. Al-Azzoni, I., Down, D.G., Khedri, R.: Modelling and verification of cryptographic protocols using coloured Petri nets and Design/CPN. *Nordic Journal of Computing* 12(3) (2005)
6. Best, E., Devillers, R., Koutny, M.: *Petri Net Algebra*. In: *EATCS Monographs on TCS*. Springer, Heidelberg (2001)
7. Best, E., Frączak, W., Hopkins, R.P., Klaudel, H., Pelz, E.: M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages. *Acta Informatica* 35 (1998)
8. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: *IEEE Symposium on Security and Privacy* (2006)
9. Bouroulet, R., Klaudel, H., Pelz, E.: A semantics of Security Protocol Language (SPL) using a class of composable high-level Petri nets. In: *ACSD 2004*. IEEE Computer Society, Los Alamitos (2004)
10. Bouroulet, R., Klaudel, H., Pelz, E.: Modelling and verification of authentication using enhanced net semantics of SPL (Security Protocol Language). In: *ACSD 2006*. IEEE Computer Society, Los Alamitos (2006)
11. Christensen, S., Hansen, N.D.: Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs. In: Ajmone Marsan, M. (ed.) *ICATPN 1993*. LNCS, vol. 691. Springer, Heidelberg (1993)

12. Clark, J., Jacob, J.: A survey of authentication protocol literature: Version 1.0 (1997), <http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz>
13. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
14. Crazzolara, F., Winskel, G.: Events in security protocols. In: ACM Conf on Computer and Communications Security. ACM Press, New York (2001)
15. Denker, G., Millen, J.: CAPSL Integrated Protocol Environment. In: DISCEX 2000. IEEE Computer Society, Los Alamitos (2000)
16. Dill, D.L., Drexler, A.J., Hu, A.J., Han Yang, C.: Protocol Verification as a Hardware Design Aid. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors (1992)
17. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29 12 (1983)
18. Donovan, B., Norris, P., Lowe, G.: Analyzing a library of security protocols using Casper and FDR. In: Workshop on Formal Methods and Security Protocols (1999)
19. Evangelista, S.: High Level Petri Nets Analysis with Helena. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 455–464. Springer, Heidelberg (2005), <http://helena.cnam.fr/>
20. Gordon, A., Jeffrey, A.: Authenticity by Typing for Security Protocols. In: IEEE Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2001)
21. Kao, I.-L., Chow, R.: An efficient and secure authentication protocol using uncertified keys. In: Operating Systems Review, vol. 29(3), ACM Press, New York (1995)
22. Kremer, S., Raskin, J.-F.: A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols. Journal of Computer Security 11(3) (2003)
23. Lowe, G.: An attack on the Needham-Schroeder public key authentication protocol. Information Processing Letters 56(3) (1995)
24. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using (FDR). In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055. Springer, Heidelberg (1996)
25. Meadows, C.: Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. IEEE Journal on Selected Areas in Communication 21(1) (2003)
26. Milner, R.: Communicating and mobile systems: The π -calculus. Cambridge University Press, Cambridge (1999)
27. Mitchell, J.C., Mitchell, M., Stern, U.: Automated Analysis of Cryptographic Protocols Using Mur ϕ . In: IEEE Symposium on Security and Privacy. IEEE Computer Society Press, Los Alamitos (1997)
28. Needham, R.M., Schroeder, M.D.: Using Encrypton for Authentication in Large Networks of Computers. Comm. of the ACM 21(12) (1978)
29. Nieh, B.B., Tavares, S.E.: Modelling and Analyzing Cryptographic Protocols Using Petri Nets. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718. Springer, Heidelberg (1993)
30. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystem. Comm. of the ACM 21(2) (1978)
31. Schneier, B.: Applied Cryptography. Wiley, Chichester (1996)
32. Thayer, F., Herzog, J.C., Guttman, J.D.: Strand Spaces: Why is a Security Protocol Correct? In: IEEE Symposium on Security and Privacy (1998)
33. Song, D.: Athena: A new efficient automatic checker for security protocol analysis. In: CSFW 1999. IEEE Computer Society Press, Los Alamitos (1999)

A Symbolic Algorithm for the Synthesis of Bounded Petri Nets^{*}

J. Carmona¹, J. Cortadella¹, M. Kishinevsky², A. Kondratyev³, L. Lavagno⁴,
and A. Yakovlev⁵

¹ Universitat Politècnica de Catalunya, Spain

² Intel Corporation, USA

³ Cadence Berkeley Laboratories, USA

⁴ Politecnico di Torino, Italy

⁵ Newcastle University, UK

Abstract. This paper presents an algorithm for the synthesis of bounded Petri nets from transition systems. A bounded Petri net is always provided in case it exists. Otherwise, the events are split into several transitions to guarantee the synthesis of a Petri net with bisimilar behavior. The algorithm uses symbolic representations of multisets of states to efficiently generate all the minimal regions. The algorithm has been implemented in a tool. Experimental results show a significant net reduction when compared with approaches for the synthesis of safe Petri nets.

1 Introduction

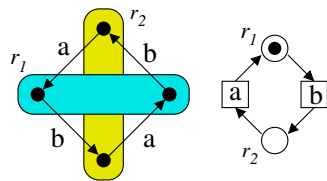
The problem of Petri net synthesis consists of building a Petri net that has a behavior equivalent to a given transition system. The problem was first addressed by Ehrenfeucht and Rozenberg [ER90] introducing *regions* to model the sets of states that characterize marked places. Mukund [Muk92] extended the concept of region to synthesize nets with weighted arcs.

Different variations of the synthesis problem have been studied in the past [Dar07]. Most of the efforts have been devoted to the decidability problem, i.e. questioning about the existence of a Petri net with a specified behavior. An exception is in [BBD95] where polynomial algorithms for the synthesis of bounded nets were presented. These methods have been implemented in the tool SYNETH [Cai02].

Desel and Reisig [DR96] reduced the synthesis problem to the calculation of the subset of *minimal regions*. In [HKT95, HKT96] the classical trace model by Mazurkiewicz [Maz87] was extended to describe the behavior of general Petri nets.

^{*} Work of J. Carmona and J. Cortadella has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation. Work of A. Yakovlev was supported by EPSRC, Grants EP/D053064/1 and EP/E044662/1.

In most of the previous approaches, two major constraints are imposed: (1) any event must be represented by *only one transition*, and (2) the reachability graph of the Petri net must be *isomorphic* to the initially given transition system. The approach presented in [CKLY98] relaxed the previous conditions. The condition of isomorphism was replaced by *bisimilarity* [Mil89] and, as a result, the classical *separation axioms* did not have to hold for pairs of bisimilar states. Additionally, the Petri net was allowed to have multiple transitions with the same label (event). This approach was only applicable to safe Petri nets. The figure shows a transition system where the separation axioms do not hold, but a Petri net with bisimilar behavior can be found with the methods described in [CKLY98].



1.1 Motivation and Contributions

There are several areas of interest for the synthesis of Petri nets. One of them is *visualization* [VPWJ07]. Understanding the behavior of large concurrent systems such as business processes [BDLS07] or asynchronous circuits [CKLY98] is a complex task that can be facilitated by visualizing the causality and concurrency relations of their events. In these cases, the non-existence of a Petri net should not be the cause that prevents visualization.

Another interesting area of application is *direct synthesis*, which consists of implementing concurrent systems by representing them as Petri nets and mapping the places and transitions into software or hardware realizations of these objects (e.g. [SBY07]). Finding succinct representations of concurrent behaviors contributes to derive efficient implementations.

On the other hand, the state spaces of such concurrent systems do not always have an explicit representation. Instead, symbolic representations are often used. This is the case of the transition relations used to verify temporal properties [CGP00] or the gate netlists used to represent circuits. The behavior is implicitly represented by the reachable states obtained from these representations.

This paper provides an efficient synthesis approach for concurrent systems. An algorithm for bounded Petri nets synthesis based on the theory of general regions is presented. Starting from the algorithms for synthesizing safe Petri Nets in [CKLY98], the theory and algorithms are extended by generalizing the notion of excitation closure from sets of states to multisets of states. The extension covers the case of the k -bounded Petri nets with weighted arcs. The paper also proposes heuristics to handle transition systems which do not satisfy the notion of excitation closure and hence cannot be modeled with general Petri nets with uniquely labelled transitions. In this case, methods for splitting events allow to generate Petri Nets with multiple occurrences of the same original label. In summary, the main features of the approach are:

- The synthesis of weighted Petri nets.
- The use of symbolic methods based on BDDs to explore large state spaces.
- Efficient heuristics for event splitting.

1.2 Two Illustrative Examples

Figure 1 depicts a finite transition system with 9 states and 3 events. After synthesis, the Petri net at the right is obtained. Each state has a 3-digit label that corresponds to the marking of places p_1 , p_2 and p_3 of the Petri net, respectively. The shadowed states represent the general region that characterizes place p_2 . Each grey tone represents a different multiplicity of the state (4 for the darkest and 1 for the lightest). Each event has a gradient with respect to the region (+2 for a , -1 for b and 0 for c). The gradient indicates how the event changes the multiplicity of the state after firing. For the same example, the equivalent *safe* Petri net generated by `petrify` [CKLY98] has 5 places and 10 transitions.

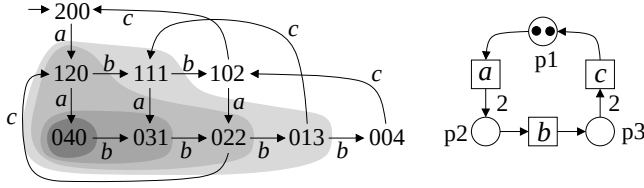


Fig. 1. A transition system and an equivalent bounded Petri net

Another example is shown in Fig. 2. The transition system models a behavior with OR-causality, i.e. the event c can be fire as soon as a or b have fired. The net model is much simpler and intuitive if bounded nets are used. Instead, the model with a safe Petri net needs to represent the events a and b with multiple transitions.

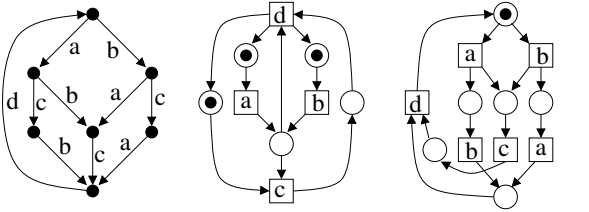


Fig. 2. (a) Transition system, (b) 2-bounded Petri net, (c) safe Petri net

The algorithm presented in this paper is based on an efficient manipulation of multisets to explore the minimal general regions of a finite transition system.

2 Background

2.1 Petri Nets and Finite Transition Systems

Definition 1. A Petri net is a tuple (P, T, W, M_0) where P and T represent finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$

is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.

Definition 2 (Transition system). A transition system (or TS) is a tuple (S, Σ, E, s_{in}) , where S is a set of states; Σ is an alphabet of actions, such that $S \cap \Sigma = \emptyset$; $E \subseteq S \times \Sigma \times S$ is a set of (labelled) transitions; and s_{in} is the initial state.

Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system. We consider connected TSs that satisfy the following axioms:

- S and E are finite sets.
- Every event has an occurrence: $\forall e \in \Sigma : \exists (s, e, s') \in E$;
- Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

A state without outgoing arcs is called *deadlock* state.

2.2 Multisets

The following definitions establish the necessary background to understand the concept of region, introduced in Section 2.3. A region is a multiset where additional conditions hold. We start by introducing the multiset terminology.

Definition 3 (Multiset). Given a set S , a multiset r of S is a mapping $r : S \rightarrow \mathbb{N}$. We will also use a set notation for multisets. For example, let $S = \{s_1, s_2, s_3, s_4\}$, then a multiset $r = \{s_1^3, s_2^2, s_3\}$ corresponds to the following mapping $r(s_1) = 3, r(s_2) = 2, r(s_3) = 1, r(s_4) = 0$.

Henceforth we will assume r , r_1 and r_2 to be multisets of a set S .

Definition 4 (Support of a multiset). The support of a multiset r is defined as

$$\text{supp}(r) = \{s \in S \mid r(s) > 0\}$$

Definition 5 (Power of a multiset). The power of a multiset r , denoted by $r^\diamond$, is defined as

$$r^\diamond = \max_{s \in S} r(s)$$

For instance, for the multiset $r = \{s_1^3, s_2^2, s_3\}$, $r^\diamond = 3$.

Definition 6 (Trivial multisets). A multiset r is said to be trivial if $r(s) = r(s')$ for all $s, s' \in S$. The trivial multisets will be denoted by $\mathbf{0}, \mathbf{1}, \dots, \mathbf{K}$ when $r(s) = 0, r(s) = 1, \dots, r(s) = k$, for every $s \in S$, respectively.

Definition 7 (k -bounded multiset). A multiset r is k -bounded if for all $s \in S : r(s) \leq k$.

Definition 8 (Union, intersection and difference of multisets). *The union, intersection and difference of two multisets r_1 and r_2 are defined as follows:*

$$\begin{aligned}(r_1 \cup r_2)(s) &= \max(r_1(s), r_2(s)) \\ (r_1 \cap r_2)(s) &= \min(r_1(s), r_2(s)) \\ (r_1 - r_2)(s) &= \max(0, r_1(s) - r_2(s))\end{aligned}$$

Definition 9 (Subset of a multiset). *A multiset r_1 is a subset of a multiset r_2 ($r_1 \subseteq r_2$) if*

$$\forall s \in S : r_1(s) \leq r_2(s)$$

As usual, we will denote by $r_1 \subset r_2$ the fact that $r_1 \subseteq r_2$ and $r_1 \neq r_2$.

Definition 10 (k -topset of a multiset). *The k -topset of a multiset r , denoted by $\top_k(r)$, is defined as follows:*

$$\top_k(r)(s) = \begin{cases} r(s) & \text{if } r(s) \geq k \\ 0 & \text{otherwise} \end{cases}$$

A multiset r_1 is a topset of r_2 if there exists some k for which $r_1 = \top_k(r_2)$.

Examples. The multiset $\{s_1^3, s_3\}$ is a subset of $\{s_1^3, s_2^2, s_3\}$, but it is not a topset. The multisets $\{s_1^3, s_2^2\}$ and $\{s_1^3\}$ are the 2- and 3-topsets of $\{s_1^3, s_2^2, s_3\}$, respectively. As it will be shown in Section 4, k -topsets are the main objects to look at when constructing the (weighted) flow relation for Petri net synthesis.

Property 1 (Partial order of multisets). The relation $\subseteq$ (subset) on the set of multisets of S is a partial order.

Property 2 (The lattice of k -bounded multisets). The set of k -bounded multisets of a set S with the relation $\subseteq$ is a lattice. The meet and join operations are the intersection and union of multisets respectively. The least and greatest elements are $\mathbf{0}$ and $\mathbf{K}$ respectively.

2.3 General Regions

Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system. In this section, we will consider multisets of the set S .

Definition 11 (Gradient of a transition). *Given a multiset r , the gradient of a transition (s, e, s') is defined as*

$$\Delta_r(s, e, s') = r(s') - r(s)$$

An event e is said to have a non-constant gradient in r if there are two transitions (s_1, e, s'_1) and (s_2, e, s'_2) such that

$$r(s'_1) - r(s_1) \neq r(s'_2) - r(s_2)$$

Definition 12 (Region). *A multiset r is a region if all events have a constant gradient in r .*

The original notion of region from [ER90] was restricted to subsets of S , i.e. events could only have gradients in $\{-1, 0, +1\}$.

Definition 13 (Gradient of an event). *Given a region r and an event e with $(s, e, s') \in E$, the gradient of e in r is defined as*

$$\Delta_r(e) = r(s') - r(s)$$

Definition 14 (Minimal region). *A region r is minimal if there is no other region $r' \neq \mathbf{0}$ such that $r' \subset r$.*

Note that the trivial region $\mathbf{0}$ is not considered to be minimal.

Theorem 1. *Let r be a region such that $\mathbf{1} \subset r$. Then r is not a minimal region.*

Proof. Trivial. A smaller region can be obtained by subtracting 1 from each $r(s)$. $\square$

Definition 15 (Excitation and switching regions¹). *The excitation region of an event e , $\text{ER}(e)$, is the set of states in which e is enabled, i.e.*

$$\text{ER}(e) = \{s \mid \exists s' : (s, e, s') \in E\}$$

The switching region of an event e , $\text{SR}(e)$, is the set of states reachable from $\text{ER}(e)$ after having fired e , i.e.

$$\text{SR}(e) = \{s \mid \exists s' : (s', e, s) \in E\}$$

For convenience, $\text{ER}(e)$ and $\text{SR}(e)$ will be also considered as multisets of states when necessary.

Definition 16 (Pre- and post-regions). *A region r is a pre-region of e if $\text{ER}(e) \subseteq r$. A region r is a post-region of e if $\text{SR}(e) \subseteq r$. The sets of pre- and post-regions of an event e are denoted by ${}^{\circ}e$ and e° respectively.*

Note that a region r can be a pre-region and a post-region of the same event in case $\text{ER}(e) \cup \text{SR}(e) \subseteq r$. The behavior modeled in this situation can be seen as a self-loop in a Petri net.

¹ Excitation and switching regions are not regions in the terms of Definition 12. They correspond to the set of states in which an event is enabled or just fired, correspondingly. The terms are used due to historical reasons.

Properties of regions

Property 3. Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system without deadlock states. Then, for any region r there is an event e for which $\Delta_r(e) \leq 0$.

Proof. By contradiction. Assume that $\Delta_r(e) > 0$ for all events. Then for all arcs (s, e, s') we have that $r(s') > r(s)$. Since TS has no deadlock states and S is finite, there is at least one cycle $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n \rightarrow s_0$. This would result in $r(s_0) > r(s_0)$, which is a contradiction. $\square$

Property 4. Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system in which there is a transition $(s, e', s_{in}) \in E$. Then, for any region r there is an event e for which $\Delta_r(e) \geq 0$.

Proof. By contradiction. Assume that $\Delta_r(e) < 0$ for all events. Then for all arcs (s, e, s') we have that $r(s') < r(s)$. Since there is $(s, e', s_{in}) \in E$, there is at least one cycle $s_{in} \rightarrow s_1 \rightarrow \dots \rightarrow s_n = s \rightarrow s_{in}$. This would result in $r(s_{in}) < r(s_{in})$, which is a contradiction. $\square$

Property 5. Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system without deadlock states. Then, any region $r \neq \emptyset$ is the pre-region of some event e .

Proof. Property 3 guarantees the existence of an event e such that $\Delta_r(e) \leq 0$. If $\Delta_r(e) < 0$, then every state in $\text{ER}(e)$ must be in the support of r and the claim holds. If every event e fulfilling Property 3 satisfies $\Delta_r(e) = 0$, then $\text{supp}(r) = S$: if the contrary is assumed, since the transition system is deadlock-free and $r \neq \emptyset$, the states in the support of r must be connected to some states out of r . However, if every event has non-negative gradient in r , then in this situation r must contain all the states. $\square$

Property 6. Let $\text{TS} = (S, \Sigma, E, s_{in})$ be a transition system. Then, any region $r \neq \emptyset$ is the pre-region or the post-region of some event e .

Proof. A similar reasoning of the proof of Property 5 can be applied here, but using Properties 3 and 4. $\square$

2.4 Excitation-Closed TSs

This section defines a specific class of transition systems, called *excitation-closed*, for which the synthesis approach presented in this paper guarantees that the Petri net obtained has a reachability graph bisimilar to the initial transition system.

Definition 17 (Enabling topset). *The set of smallest enabling topsets of an event e is denoted by $\star e$ and defined as follows:*

$$\star e = \{q \mid \exists r \in {}^\circ e, k > 0 : q = \top_k(r) \wedge \text{ER}(e) \subseteq \top_k(r) \wedge \text{ER}(e) \not\subseteq \top_{k+1}(r)\}$$

Intuitively, q belongs to $\star e$ if it is the topset of a pre-region r of e and there is no larger topset that includes $\text{ER}(e)$.

Definition 18 (ECTS). A TS is excitation closed if it satisfies the following two properties:

1. *Excitation closure.* For each event e

$$\bigcap_{q \in \star e} \text{supp}(q) = \text{ER}(e)$$

2. *Event effectiveness.* For each event e , $\circ e \neq \emptyset$

3 Generation of Minimal Regions

Now we are ready to describe an algorithm to generate the set of all k -bounded minimal regions from a given transition system. Informally, the generation of minimal regions is based on Property 6, that states that any region is either a pre-region or post-region of some event. Therefore the exploration of regions starts by considering the ER and SR of every event, and expands those sets that violate the region condition. Provided that only minimal expansions are considered at each step of the algorithm, the generation of all the minimal regions is guaranteed. Hence the notion of multiset expansion is crucial in this paper.

Multisets are expanded with the aim of ensuring constant gradient for every event. Formally, given a multiset r and an event e with non-constant gradient, the following definitions characterize the set of regions that include r .

Definition 19. Let $r \neq \mathbf{0}$ be a multiset. We define

$$\begin{aligned} \mathcal{R}_g(r, e) &= \{r' \supseteq r \mid r' \text{ is a region and } \Delta_{r'}(e) \leq g\} \\ \mathcal{R}^g(r, e) &= \{r' \supseteq r \mid r' \text{ is a region and } \Delta_{r'}(e) \geq g\} \end{aligned}$$

$\mathcal{R}_g(r, e)$ is the set of all regions larger than r in which the gradient of e is smaller than or equal to g . Similar for $\mathcal{R}^g(r, e)$ and the gradient of e greater than or equal to g . Notice that in this definition a gradient g is used to partition the set of regions including r into two classes. This binary partition is the basis for the calculation of minimal k -bounded regions that will be presented at the end of this section.

To expand a multiset in order to convert it into a region, it is necessary to know the lower bound needed for the increase in each state in order to satisfy a given gradient constraint. The next functions provide these lower bounds:

Definition 20. Given a multiset r , a state s and an event e , the following δ functions are defined²:

$$\begin{aligned} \delta_g(r, e, s) &= \max(0, \max_{(s, e, s') \in E} (r(s') - r(s) - g)) \\ \delta^g(r, e, s) &= \max(0, \max_{(s', e, s) \in E} (r(s') - r(s) + g)) \end{aligned}$$

² For convenience, we consider $\max_{x \in D} P(x) = 0$ when the domain D is empty.

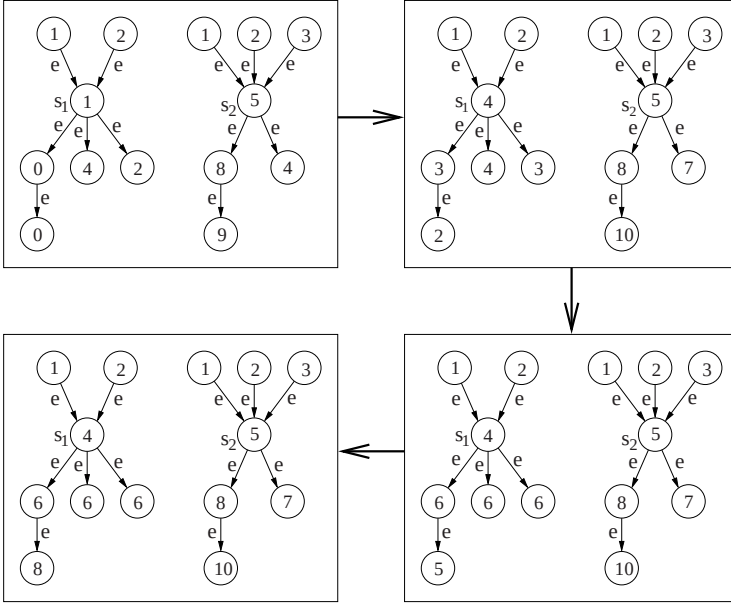


Fig. 3. Successive calculations of $\sqcap^2(r, e)$

Informally, δ_g denotes a lower bound for the increase of $r(s)$, taking into account the arcs leaving from s , to force $\Delta_{r'}(e) \leq g$ in some region r' larger than r . Similarly, δ^g denotes a lower bound taking into account the arcs arriving at s , to force $\Delta_{r'}(e) \geq g$. Let us use Figure 3 to illustrate this concept. In the figure each state is labeled with $r(s)$. For the states s_1 and s_2 in the top-left figure, we have:

$$\delta^2(r, e, s_1) = 3 \quad \delta^2(r, e, s_2) = 0$$

$\delta^2(r, e, s_1)$ is determined by the arc $\boxed{2} \xrightarrow{e} \boxed{1}$ and indicates that $r'(s_1) \geq 4$ in case we seek a region $r' \supset r$ with $\Delta_{r'}(e) \geq 2$. Analogously, Figure 4 illustrates the symmetrical concept. For the states s_1 and s_2 in the leftmost figure we have:

$$\delta_1(r, e, s_1) = 2 \quad \delta_1(r, e, s_2) = 2$$

$\delta_1(r, e, s_1)$ is determined by the arc $\boxed{1} \xrightarrow{e} \boxed{4}$, indicating that $r'(s_1) \geq 3$ for $\Delta_{r'}(e) \leq 1$. Similarly, $\delta_1(r, e, s_2)$ is determined by the arc $\boxed{5} \xrightarrow{e} \boxed{8}$.

Definition 21. Given a multiset r and an event e , the multisets $\sqcap_g(r, e)$ and $\sqcap^g(r, e)$ are defined as follows:

$$\begin{aligned} \sqcap_g(r, e)(s) &= r(s) + \delta_g(r, e, s) \\ \sqcap^g(r, e)(s) &= r(s) + \delta^g(r, e, s) \end{aligned}$$

Intuitively, $\sqcap_g(r, e)$ is a safe move towards growing r and obtaining all regions r' with $\Delta_r(e) \leq g$. Similarly, $\sqcap^g(r, e)$ for those regions with $\Delta_r(e) \geq g$. It is

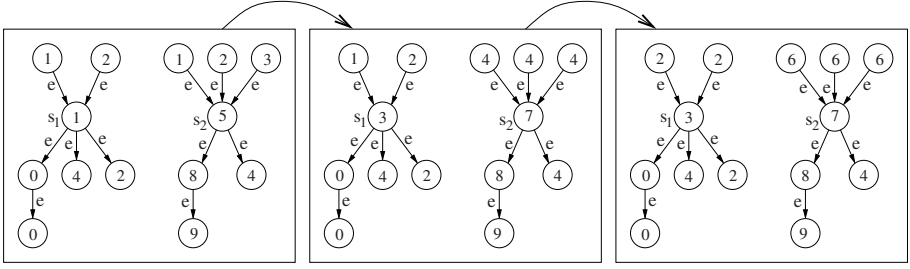


Fig. 4. Successive calculations of $\Pi_1(r, e)$

easy to see that $\Pi_g(r, e)$ and $\Pi^g(r, e)$ always derive multisets larger than r . The successive calculations of $\Pi^g(r, e)$ and $\Pi_g(r, e)$ are illustrated in Figures 3 and 4, respectively.

Theorem 2 (Expansion on events)

- (a) Let $r \neq \mathbf{0}$ be a multiset and e an event such that there exists some (s, e, s') with $r(s') - r(s) > g$. The following hold:
1. $r \subset \Pi_g(r, e)$
 2. $\mathcal{R}_g(r, e) = \mathcal{R}_g(\Pi_g(r, e), e)$
- (b) Let $r \neq \mathbf{0}$ be a multiset and e an event such that there exists some (s, e, s') with $r(s') - r(s) < g$. The following hold:
1. $r \subset \Pi^g(r, e)$
 2. $\mathcal{R}^g(r, e) = \mathcal{R}^g(\Pi^g(r, e), e)$

Proof (We prove item (a), item (b) is similar.)

(a.1) Given the event e , for all states s, s' with (s, e, s') either (i) $r(s') - r(s) \leq g$ or (ii) $r(s') - r(s) > g$. In situation (i), the following equality holds: $r(s) = \Pi_g(r, e)(s)$ because $\delta_g(r, e, s) = 0$ by Definition 20. In situation (ii), $\delta_g(r, e, s) > 0$, and therefore $r(s) < \Pi_g(r, e)(s)$. Given that the rest of states without outgoing arcs labeled e fulfill also $r(s) = \Pi_g(r, e)(s)$, and because there exists at least one transition satisfying (ii), the claim holds.

(a.2) To obtain $\mathcal{R}_g(r, e)$ it is necessary to guarantee $\Delta_{r'}(e) \leq g$ for each region $r' \supseteq r$. Given (s, e, s') with $r(s') - r(s) > g$, two possibilities can induce a gradient lower than g , e.g. decreasing $r(s')$ or increasing $r(s)$, but only the latter leads to a multiset with r as a subset. $\square$

Figure 5 presents an algorithm for the calculation of all minimal k -bounded regions. It is based on a dynamic programming approach that, starting from a multiset, generates an exploration tree in which an event with non-constant gradient is chosen at each node. All possible gradients for that event are explored by means of a binary search. Dynamic programming with memoization avoids the exploration of multiple instances of the same node. The final step of the algorithm (lines 16–17) removes all those multisets that are neither regions nor minimal regions that have been generated during the exploration.

```

generate_minimal_regions (TS,k) {
1:   $R = \emptyset$ ; /* set of explored multisets */
2:   $P = \{ER(e) \mid e \in E\} \cup \{SR(e) \mid e \in E\}$ ;
3:  while ( $P \neq \emptyset$ ) /* multisets pending for exploration */
4:     $r = \text{remove\_one\_element}(P)$ ;
5:    if ( $r \notin R$ ) /* dynamic programming with memoization */
6:       $R = R \cup \{r\}$ ;
7:      if ( $r$  is not a region)
8:         $e = \text{choose\_event\_with\_non\_constant\_gradient}(r)$ ;
9:         $(g_{min}, g_{max}) = \text{“minimum and maximum gradients of } e \text{ in } r\text{”}$ ;
10:        $g = \lfloor (g_{min} + g_{max})/2 \rfloor$ ; /* gradient for binary search */;
11:        $r_1 = \sqcap_g(r, e)$ ; if ( $(r_1^\diamond \leq k) \wedge (1 \not\subset r_1)$ )  $P = P \cup \{r_1\}$  endif;
12:        $r_2 = \sqcap^{g+1}(r, e)$ ; if ( $(r_2^\diamond \leq k) \wedge (1 \not\subset r_2)$ )  $P = P \cup \{r_2\}$  endif;
13:     endif
14:   endif
15: endwhile;
16:  $R = R \setminus \{r \mid r \text{ is not a region}\}$ ; /* Keep only regions */
17:  $R = R \setminus \{r \mid \exists r' \in R : r' \subset r\}$ ; /* Keep only minimal regions */
}

```

Fig. 5. Algorithm for the generation of all k -bounded minimal regions

Theorem 3. *The algorithm generate_minimal_regions in Figure 5 calculates all k -bounded minimal regions.*

Proof. The proof is based on the following facts:

1. All minimal regions are a pre- or a post-region of some event (property 6). Any pre- (post-) region of an event is larger than its ER (SR). Line 2 of the algorithm puts all seeds for exploration in P . These seeds are the ERs and SRs of all events.
2. Each r that is not a region is enlarged by $\sqcap_g(r, e)$ and $\sqcap^{g+1}(r, e)$ for some event e with non-constant gradient. Given that $g = \lfloor (g_{min} + g_{max})/2 \rfloor$, there is always some transition $s_1 \xrightarrow{e} s_2$ such that $r(s_2) - r(s_1) = g_{max} > g$ and some transition $s_3 \xrightarrow{e} s_4$ such that $r(s_4) - r(s_3) = g_{min} < g + 1$. Therefore, the conditions for theorem 2 hold. By exploring $\sqcap_g(r, e)$ and $\sqcap^{g+1}(r, e)$, no minimal regions are missed.
3. The algorithm halts since the set of k -bounded multisets with $\subseteq$ is a lattice and the multisets derived at each level of the tree are larger than their predecessors. Thus, the exploration will halt at those nodes in which the power of the multiset is larger than k (lines 11–12). The condition $(1 \not\subset r')$ in lines 11–12 improves the efficiency of the search (see theorem 1). $\square$

For the case of safe Petri nets, line 9 of the algorithm always gives $g = g_{min} = 0$ and $g_{max} = 1$. The calculation of $\sqcap_0(r, e)$ and $\sqcap^1(r, e)$ with the constraints $r_1^\diamond, r_2^\diamond \leq 1$ is equivalent to the expansion of sets of states presented in lemma 4.2 of [CKLY98].

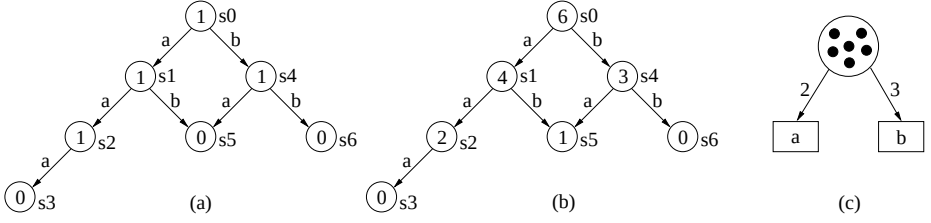


Fig. 6. Generation of minimal region: (a) $ER(a)$, (b) final region after the exploration shown in table 1, (c) equivalent Petri net

Table 1. Path in the exploration tree for the generation of the region in Figure 6(b)

r_i	$r_i(s)$							illegal	chosen			
	s_0	s_1	s_2	s_3	s_4	s_5	s_6	events	event	g_{min}	g_{max}	g
$r_1 = ER(a)$	1	1	1	0	1	0	0	$\{a, b\}$	a	-1	0	-1
$r_2 = \sqcap_{-1}(r_1, a)$	2	2	1	0	1	0	0	$\{a, b\}$	b	-2	-1	-2
$r_3 = \sqcap_{-2}(r_2, b)$	3	2	1	0	2	0	0	$\{a, b\}$	a	-2	-1	-2
$r_4 = \sqcap_{-2}(r_3, a)$	4	3	2	0	2	0	0	$\{a, b\}$	b	-3	-2	-3
$r_5 = \sqcap_{-3}(r_4, b)$	5	3	2	0	3	0	0	$\{a, b\}$	b	-3	-2	-3
$r_6 = \sqcap_{-3}(r_5, b)$	6	3	2	0	3	0	0	$\{a\}$	a	-3	-1	-2
$r_7 = \sqcap_{-2}(r_6, a)$	6	4	2	0	3	1	0	$\emptyset$				

Figure 6 presents an example of calculation of minimal regions. Starting from $ER(a)$, the multiset is iteratively enlarged until a minimal region is obtained. Table 1 describes one of the paths of the exploration tree.

3.1 Symbolic Representation of Multisets

All the operations required in the algorithm of Figure 5 to manipulate the multisets can be efficiently implemented by using a symbolic representation. A multiset can be modeled as a vector of Boolean functions, where the function at position i describes the characteristic function of the set of states having cardinality i . Hence, multiset operations (union, intersection and complement) can be performed as logic operations (disjunction, conjunction and complement) on Boolean functions. An array of Binary Decision Diagrams (BDDs) [Bry86] is used to represent implicitly a multiset.

4 Synthesis of Petri Nets

The synthesis of Petri nets can be based on the generation of minimal regions described by the algorithm in Figure 5.

For the sake of efficiency, different strategies can be sought for synthesis. They can differ in the way the exploration tree is built. One of the possible strategies would consist in defining upper bounds on the capacity of the regions (k_{max})

and on the gradient of the events (w_{max}). The tree can then be explored without surpassing these bounds. For example, one could start with $k_{max} = g_{max} = 1$ to check whether a safe Petri net can be derived. In case the excitation closure does not hold, the bounds could be increased, etc. Splitting labels could be done when the bounds go too high without finding the excitation closure.

By tuning the search algorithm with different parameters, the search for minimal regions can be pruned at the convenience of the user.

Once all minimal regions have been generated, excitation closure must be verified (see Definition 18). In case excitation closure holds, a minimal *saturated*³ PN can be generated as follows:

- For each event e , a transition labeled with e is generated.
- For each minimal region r_i , a place p_i is generated.
- Place p_i contains k tokens in the initial marking if $r_i(s_{in}) = k$.
- For each event e and minimal region r_i such that $r_i \in {}^\circ e$ find $q \in {}^\star e$ such that $q = \top_k(r_i)$. Add an arc from place p_i to transition e with weight k . In case $\Delta_{r_i}(e) > -k$ add an arc from transition e to place p_i with weight $k + \Delta_{r_i}(e)$.
- For each event e and minimal region r_i such that $SR(e) \subseteq r_i$ add an arc from transition e to place p_i with weight $\Delta_{r_i}(e)$ ⁴.

Given that the approach presented in this paper is a generalization for the case of safe Petri nets from [CKLY98], the following theorem can be proved:

Theorem 4. *Let TS be an excitation closed transitions system. The synthesis approach of this section derives a PN with reachability graph bisimilar to TS.*

Proof. In [CKLY98], a proof for the case of safe Petri nets was given (Theorem 3.4). The proof for the bounded case is a simple extension, where only the definition of excitation closure must be adapted to deal with multisets. Due to the lack of space we sketch the steps to attain the generalization.

The proof shows, by induction on the length of the traces leading to a state, that there is a correspondence between the reachability graph of the synthesized net and TS. The crucial idea is that non-minimal regions can be removed from a state in TS to derive a state in the reachability graph of the synthesized net. Moreover excitation closure ensures that this correspondence preserves the transitions in both transition systems. Based on this correspondence, a bisimulation is defined between the states of TS and the states of the reachability graph of the synthesized net. $\square$

The algorithm described in this section is complete in the sense that if excitation closure holds in the initial transition system and a bound large enough is used, the computation of a set of minimal regions to derive a Petri net with bisimilar behavior is guaranteed.

³ The net synthesized by the algorithm is called saturated since all regions are mapped into the corresponding places [CKLY98].

⁴ If $SR(e) \subseteq r_i$, then the Definition 13 makes $\Delta_{r_i}(e) \geq 0$.

4.1 Irredundant Petri Nets

A minimal saturated PN can be redundant. There are two types of redundancies that can be considered in a minimal saturated PN:

1. A minimal region is not necessary to guarantee the excitation closure of any of the events for which the ER is included in it. In this case, the corresponding place can be simply removed from the PN.
2. Given a minimal region r , its corresponding place p and an event e such that $\text{ER}(e) \subseteq \top_k(r)$ and $\Delta_r(e) = g$, if the excitation closure can still be ensured for e by considering $\top_{k-1}(r)$ instead of $\top_k(r)$, then the arcs $p \xrightarrow{k} e$ and $e \xrightarrow{k+g} p$ in the PN can be substituted by the arcs $p \xrightarrow{k-1} e$ and $e \xrightarrow{k+g-1} p$ respectively as long as $k + g - 1 \geq 0$. In the case that $k + g - 1 = 0$, the arc $e \rightarrow p$ can be removed also.

In fact, the first case of redundancy can be considered as a particular case of the second. If we consider that $\top_0(r) = S$, we can say that a region r is redundant when we can ensure the excitation closure of all events by using always $\top_0(r)$ instead of $\top_k(r)$ for some $k > 0$. Note that in this case, the region would be represented by an isolated place (all arcs have weight 0) that could be simply removed from the Petri net without changing its behavior.

5 Splitting Events

Splitting events is necessary when the excitation closure does not hold. When an event is split into different copies this corresponds to different transitions in the Petri net with the same label. This section presents some heuristics to split events.

5.1 Splitting Disconnected ERs

Assume that we have a situation such as the one depicted in Figure 7 in which

$$EC(e) = \bigcap_{q \in \star e} \text{supp}(q) \neq \text{ER}(e)$$

However, $\text{ER}(e)$ has several disconnected components $\text{ER}_i(e)$. In the figure, the white part in a $\text{ER}_i(e)$ represents those states in $EC(e) - \text{ER}_i(e)$. For some of those components, $EC(e)$ has no states adjacent to $\text{ER}_i(e)$ ($\text{ER}_2(e)$ in the Figure). By splitting event e into two events e_1 and e_2 in such a way that e_2 corresponds to ERs with no adjacent states in $EC(e)$, we will ensure at least the excitation closure of e_2 .

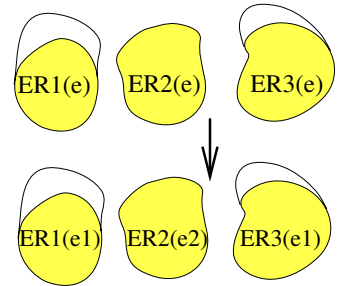


Fig. 7. Disconnected ERs

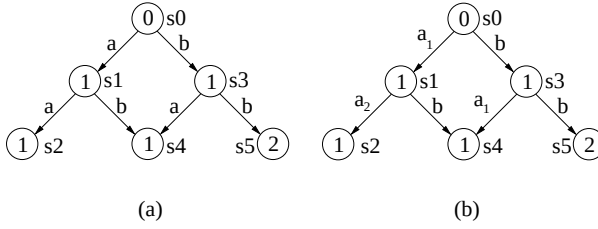


Fig. 8. Splitting on different gradients

5.2 Splitting on the Most Promising Expansion of an ER

The algorithm presented in Section 3 for generating minimal regions explores all the expansions $\sqcap_k$ and $\sqcap^k$ of an ER. When excitation closure does not hold, all these expansions are stored. Finally, given an event without excitation closure, the expansion r containing the maximum number of events with constant gradient (i.e. the expansion where less effort is needed to transform it into a region by splitting) is selected as source of splitting.

Given the selected expansion r where some events have non-constant gradient, let $|\Delta_r(e)|$ represent the number of different gradients for event e in r . The event a with minimal $|\Delta_r(a)|$ is selected for splitting. Let $g_1, g_2, \dots, g_n$ be the different gradients for a in r . Event a is split into n different events, one for each gradient g_i . Let us use the expansion depicted in Figure 8(a) to illustrate this process. In this multiset, events a and b have both non-constant gradient. The gradients for event a are $\{0, +1\}$ whereas the gradients for b are $\{-1, 0, +1\}$. Therefore event a is selected for splitting. The new events created correspond to the following ERs (shown in Figure 8(b)):

$$\begin{aligned} \text{ER}(a_1) &= \{s_0\} \\ \text{ER}(a_2) &= \{s_1, s_3\} \end{aligned}$$

Intuitively, the splitting of the event with minimal $|\Delta_r(e)|$ represents the minimal necessary legalization of some illegal event in r in order to convert it into a region.

6 Synthesis Examples

This section contains some examples of synthesis, illustrating the power of the synthesis algorithm developed in this paper. We show three examples, all them with non-safe behavior, and describe intermediate solutions that combine splitting with k -bounded synthesis.

6.1 Example 1

The example is depicted in Figure 9. If 2-bounded synthesis is applied, three minimal regions are sufficient for the excitation closure of the transition system:

$$p_0 = \{s_0^2, s_1, s_3\}, \quad p_1 = \{s_0, s_1, s_2\}, \quad p_2 = \{s_1, s_2^2, s_4\}$$

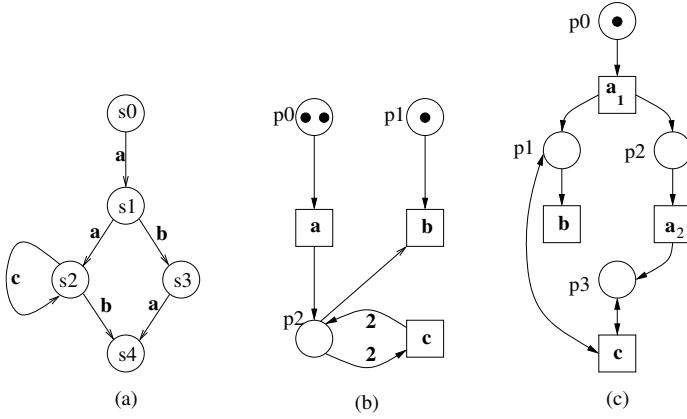


Fig. 9. (a) transition system, (b) 2-bounded Petri net, (c) safe Petri net

In this example, the excitation closure of event c is guaranteed by the intersection of the multisets $\top_1(p_1)$ and $\top_2(p_2)$, both including $ER(c)$. However, $\top_1(p_1)$ is redundant and the self-loop arc between p_1 and c can be omitted. The Petri net obtained is shown in Figure 9(b). If safe synthesis is applied, event a must be splitted. The four regions guaranteeing the excitation closure are:

$$p_0 = \{s_0\}, \quad p_1 = \{s_1, s_2\}, \quad p_2 = \{s_1, s_3\}, \quad p_3 = \{s_2, s_4\}$$

The safe Petri net synthesized is shown in Figure 9(c).

6.2 Example 2

The example is shown in Figure 10. In this case, two minimal regions are sufficient to guarantee the excitation closure when the maximal bound allowed is three (the corresponding Petri net is shown in Figure 10(b)):

$$p_0 = \{s_0^3, s_1^2, s_2, s_3\}, \quad p_1 = \{s_1, s_2^2, s_3, s_4^3, s_5^2, s_6\}$$

The excitation closure of event b is guaranteed by $\top_2(p_1)$. However we also have that $\Delta_b(p_1) = -1$, thus requiring an arc from p_1 to b with weight 2 and another arc from b to p_1 with weight 1. The former is necessary to ensure that b will not fire unless two tokens are held in p_1 . The latter is necessary to ensure the gradient -1 of b with regard to p_1 . Figures 10(c)-(d) contain the synthesis for bound 2 and 1, respectively.

6.3 Example 3

This example is depicted in Figure 11. Using bound 4, the minimal regions are the following:

$$p_0 = \{s_0\}, \quad p_1 = \{s_1^4, s_2^3, s_3, s_4^2, s_6\}, \quad p_2 = \{s_2, s_4^2, s_5, s_6^3, s_7^4\}$$

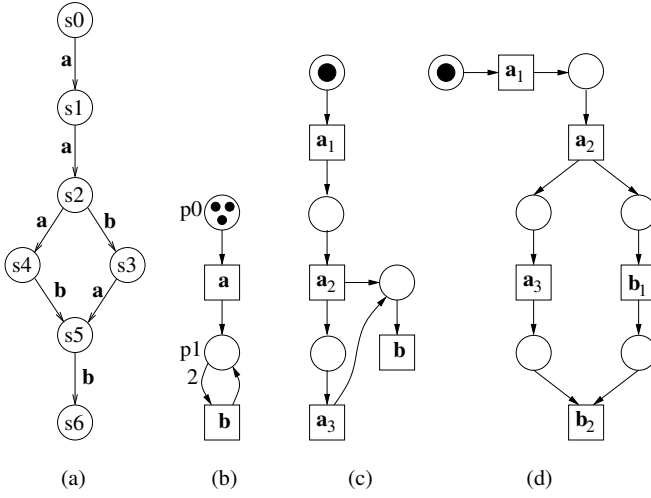


Fig. 10. (a) transition system, (b) 3-bounded, (c) 2-bounded and (d) safe Petri net

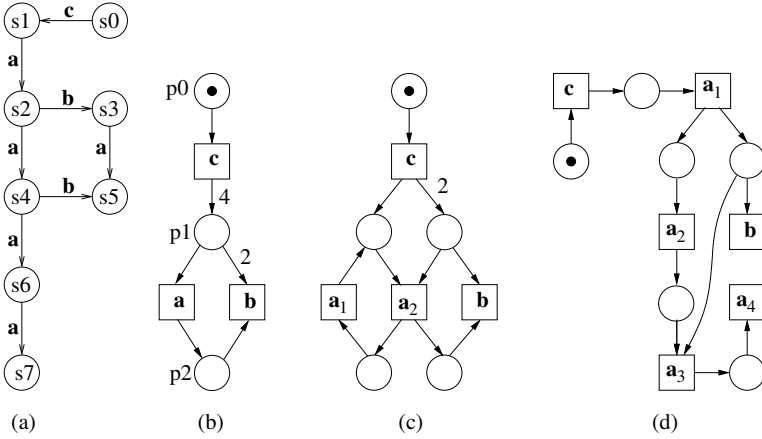


Fig. 11. (a) transition system, (b) 4-bounded, (c) 3-bounded/2-bounded and (d) safe Petri net

The synthesis with different bounds is shown in Figures 11(c)-(d). Notice that the synthesis for bounds two and three derives the same Petri net.

7 Experimental Results

In this section a set of parameterizable benchmarks are synthesized using the methods described in this paper. The following examples have been artificially created:

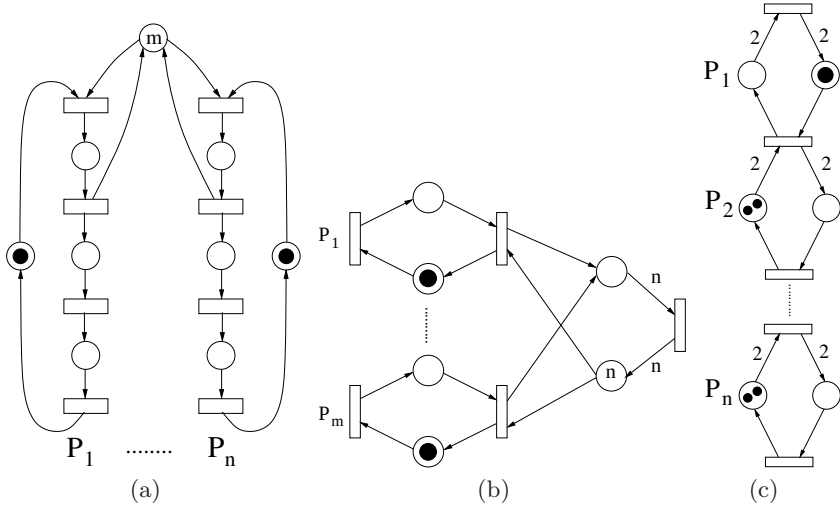


Fig. 12. Parameterized benchmarks: (a) n processes competing for m shared resources, (b) m producers and n consumers, (c) a 2-bounded pipeline of n processes

Table 2. Synthesis of parameterized benchmarks

benchmark	$ S $	$ E $	Petrify			Genet		
			$ P $	$ T $	CPU	$ P $	$ T $	CPU
SHAREDRESOURCE(3,2)	63	186	15	16	0s	13	12	0s
SHAREDRESOURCE(4,2)	243	936	20	24	5s	17	16	21s
SHAREDRESOURCE(5,2)	918	4320	48	197	180m	24	20	6m
SHAREDRESOURCE(4,3)	255	1016	21	26	2s	17	16	4m40s
PRODUCERCONSUMER(3,2)	24	68	9	10	0s	8	7	0s
PRODUCERCONSUMER(4,2)	48	176	11	13	0s	10	9	0s
PRODUCERCONSUMER(3,3)	32	92	10	13	0s	8	7	5s
PRODUCERCONSUMER(4,3)	64	240	12	17	1s	10	9	37s
PRODUCERCONSUMER(6,3)	256	1408	16	25	25s	14	13	16m
BOUNDEDPIPELINE(4)	81	135	14	9	0s	8	5	1s
BOUNDEDPIPELINE(5)	243	459	17	11	1s	10	6	19s
BOUNDEDPIPELINE(6)	729	1539	27	19	6s	12	7	5m
BOUNDEDPIPELINE(7)	2187	5103	83	68	110m	14	8	88m

1. A model for n processes competing for m shared resources, where $n > m$. Figure 12(a) describes a Petri net for this model⁵,
2. A model for m producers and n consumers, where $m > n$. Figure 12(b) describes a Petri net for this model.
3. A 2-bounded pipeline of n processes. Figure 12(c) describes a Petri net for this model.

⁵ A simplified version of this model was also synthesized by SYNET [Cai02] in [BD98].

Table 2 contains a comparison between a synthesis algorithm of safe Petri nets [CKLY98], implemented in the tool **petrify**, and the synthesis of general Petri nets as described in this paper, implemented in the prototype tool **Genet**. For each benchmark, the size of the transition system (states and arcs), number of places and transitions and cpu time is shown for the two approaches. The transition system has been initially generated from the Petri nets. Clearly, the methods developed in this paper generalize those of the tool **petrify**, and particularly the generation of minimal regions for arbitrary bounds has significantly more complexity than its safe counterpart. However, many of the implementation heuristics and optimizations included in **petrify** must also be extended and adapted to **Genet**. Provided that this optimization stage is under development in **Genet**, we concentrate on the synthesis of small examples. Hence, cpu times are only preliminary and may be improved after the optimization of the tool.

The main message from Table 2 is the expressive power of the approach developed in this paper to derive an event-based representation of a state-based one, with minimal size. If the initial transition system is excitation closed, using a bound large enough one can guarantee no splitting and therefore the number of events in the synthesized Petri net is equal to the number of different events in the transition system. Note that the excitation closure holds for all the benchmarks considered in Table 2, because the transition systems considered are derived from the corresponding Petri nets shown in Figure 12.

Label splitting is a key method to ensure the excitation closure. However, its application can degrade the solution significantly (both in terms of cpu time required for the synthesis and the quality of the solution obtained), specially if many splittings must be performed to achieve excitation closure, as it can be seen in the benchmarks **SHAREDRESOURCE(5,2)** and **BOUNDEDPIPELINE(7)**. In these examples, the synthesis performed by **petrify** derives a safe Petri net with one order of magnitude more transitions than the bounded synthesis method presented in this paper. Hence label splitting might be relegated to situations where the excitation closure does not hold (see the examples of Section 6), or when the maximal bound is not known, or when constraints are imposed on the bound of the resulting Petri net.

8 Conclusions

An algorithm for the synthesis of k -bounded Petri nets has been presented. By heuristically splitting events into multiple transitions, the algorithm always guarantees a visualization object. Still, a minimal Petri net with bisimilar behavior is obtained when the original transition systems is excitation closed.

The theory presented in this paper is accompanied with a tool that provides a practical visualization engine for concurrent behaviors.

References

- [BBD95] Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) TAPSOFT 1995. LNCS, vol. 915, pp. 364–383. Springer, Heidelberg (1995)
- [BD98] Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
- [BDLS07] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
- [Bry86] Bryant, R.: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computer-Aided Design* 35(8), 677–691 (1986)
- [Cai02] Caillaud, B.: Synet: A synthesizer of distributable bounded Petri-nets from finite automata (2002), <http://www.irisa.fr/s4/tools/synet/>
- [CGP00] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (2000)
- [CKLY98] Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers* 47(8), 859–882 (1998)
- [Dar07] Darondeau, P.: Synthesis and control of asynchronous and distributed systems. In: Basten, T., Juhás, G., Shukla, S.K. (eds.) ACSD. IEEE Computer Society, Los Alamitos (2007)
- [DR96] Desel, J., Reisig, W.: The synthesis problem of Petri nets. *Acta Informatica* 33(4), 297–315 (1996)
- [ER90] Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. *Acta Informatica* 27, 315–368 (1990)
- [HKT95] Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: A trace semantics for petri nets. *Inf. Comput.* 117(1), 98–114 (1995)
- [HKT96] Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general petri nets. *Theor. Comput. Sci.* 153(1&2), 129–170 (1996)
- [Maz87] Mazurkiewicz, A.W.: Trace theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 279–324. Springer, Heidelberg (1987)
- [Mil89] Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
- [Muk92] Mukund, M.: Petri nets and step transition systems. *Int. Journal of Foundations of Computer Science* 3(4), 443–478 (1992)
- [SBY07] Sokolov, D., Bystrov, A., Yakovlev, A.: Direct mapping of low-latency asynchronous controllers from STGs. *IEEE Transactions on Computer-Aided Design* 26(6), 993–1009 (2007)
- [VPWJ07] Verbeek, H.M.W., Pretorius, A.J., van der Aalst, W.M.P., van Wijk, J.J.: On Petri-net synthesis and attribute-based visualization. In: Proc. Workshop on Petri Nets and Software Engineering (PNSE 2007), pp. 127–141 (June 2007)

Synthesis of Nets with Step Firing Policies

Philippe Darondeau¹, Maciej Koutny², Marta Pietkiewicz-Koutny²,
and Alex Yakovlev³

¹ IRISA, campus de Beaulieu
F-35042 Rennes Cedex, France
darondeau@irisa.fr

² School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
{maciej.koutny,marta.koutny}@newcastle.ac.uk

³ School of Electrical, Electronic and Computer Engineering, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
alex.yakovlev@newcastle.ac.uk

Abstract. The unconstrained step semantics of Petri nets is impractical for simulating and modelling applications. In the past, this inadequacy has been alleviated by introducing various flavours of maximally concurrent semantics, as well as priority orders. In this paper, we introduce a general way of controlling step semantics of Petri nets through step firing policies that restrict the concurrent behaviour of Petri nets and so improve their execution and modelling features. In a nutshell, a step firing policy disables at each marking a subset of enabled steps which could otherwise be executed. We discuss various examples of step firing policies and then investigate the synthesis problem for Petri nets controlled by such policies. Using generalised regions of step transition systems, we provide an axiomatic characterisation of those transition systems which can be realised as reachability graphs of Petri nets controlled by a given step firing policy. We also provide a decision and synthesis algorithm for PT-nets and step firing policies based on linear rewards of steps, where fixing the reward of elementary transitions is part of the synthesis problem. The simplicity of the algorithm supports our claim that the proposed approach is practical.

Keywords: Petri nets, step firing policy, step transition system, regions, synthesis problem.

1 Introduction

Concurrent behaviours of Petri nets appear when the sequential firing rule is replaced with the step firing rule, according to which several transitions and several instances of a transition may be fired in a single step. The sequential reachability graph of a net is then replaced with its step reachability graph, a particular case of the step transition systems studied in [17].

Step sequence semantics occupies a middle ground in the semantical spectrum of models which have been investigated in the area of Petri nets and other models of concurrency. It is more realistic than the easy-to-define totally sequential

semantics by providing a record of concurrent execution of different actions, but is less precise than the much more complicated (and often not entirely clear) causality semantics. However, the unconstrained step semantics is both impractical and coarse from the point of view of the simulation and modelling applications. It is impractical because it suffers from an exponential explosion of possible steps enabled at a given marking. It is coarse since it does not directly capture many relevant behavioural features.

In the past, shortcomings of step semantics have in part been addressed by introducing various flavours of the maximally concurrent semantics, as well as priority orders. A typical example of taming the explosion of possible steps can be found in tools like INA [12] where only single or maximal steps can be selected for simulation. Moreover, the maximal step firing rule was used in [20] to model the behaviour of timed nets, while the locally maximal step firing rule was introduced in [13] (and the corresponding synthesis problem investigated in [14,15]) to model membrane systems and, in particular, to capture a forced synchronisation among the active reactions implied by the laws of biochemistry. Another area where a modified step firing rule is relevant is the modelling of various features and capabilities of electronic devices (as well as feedback control systems) stemming from, e.g., specific sampling time and rate. In fact, certain subtleties of concurrent systems cannot be formulated just with nets; for example, policies are indispensable to model and analyse priority arbiters (in general, resource-based view is not enough to capture synchronous features).

In our view, Petri nets with step semantics are a useful modelling approach as they are much more expressive and faithful (regarding actual concurrency) to the actual behaviour than nets with purely sequential semantics. Step semantics are also more practical than partial orders since the latter are complicated in handling (for example, it is relatively straightforward to equip a process algebra with a step sequence semantics, while doing the same for a partial order semantics is much more difficult).

In a way, step semantics is a compromise between practicality of sequential semantics and expressiveness of partial order semantics. Unfortunately, at the present moment the compromise cannot be fully realised due to the combinatorial explosion of possible steps and the lack of support for a fine tuning of behaviours. *The first concern of this paper* is to address this problem by introducing a general approach in which control is added to master the combinatorial explosion of choices. This approach will be called *step firing policies* (there is here a similarity with control policies). In short, we want to restrict the set of *resource enabled* steps in a given marking of a Petri net (i.e., steps compatible with the contents of places in this marking), using local criteria to select some steps and to *control disable* the others. Moreover, we want to define in one go a general control mechanism that applies to all possible net classes. Therefore, in the first part of the paper, we present a generic concept of τ -nets where the parameter τ is a (step) transition system whose states and labelled arcs encode all possible place markings, all possible relationships between places and transitions, and step enabledness (token free nets with a similar parametric definition were proposed

in [16]). The firing rule of τ -nets captures *resource enabledness* of steps, and their semantics is the unconstrained or ‘free’ step semantics. Step firing policies will be defined on top of τ -nets, resulting in a step semantics more restrictive than the free step semantics. A crucial property of the policies we consider is that a step can be disabled only by concurrent steps enabled at the same marking. This reflects the idea that the choices made should not depend on the history, nor on the possible futures. A particularly simple example of policies are those induced by arbitrary preorders $\preceq$ on steps. Each such firing policy consists in control disabling at each marking all steps that are not maximal w.r.t. $\preceq$ among the resource enabled steps. Such policies take inspiration from and cover the locally maximal firing rule from [13]. However, we shall also consider practically relevant step firing policies which cannot be captured by preorders. It should be said that policies impose in general a centralised control (particularly suited for regulating synchronous aspects of concurrent behaviours), which goes against the usual Petri net philosophy. Therefore, such policies should be defined and used only when necessary, e.g., in the context of simulation and control or for the modelling of electronic or biochemical systems. With this proviso, step firing policies may bridge Petri nets and various fields of application in engineering.

The second concern of this paper is to show that nets with step firing policies are amenable to technical study. We do this by studying the synthesis of τ -nets with step firing policies. In order to be able to state our contribution, let us recall the context. The basic Petri net *synthesis problem* consists in finding whether a given transition system may be realised by a Petri net with injectively labeled transitions. According to classes of nets and to behaviours defined for these nets, many versions of this problem have been dealt with, after seminal work by Ehrenfeucht and Rozenberg [10,11]. If one considers sequential behaviours of nets, a transition system is realised by a net *iff* it is isomorphic to the sequential reachability graph (or case graph) of this net. Ehrenfeucht and Rozenberg investigated the realisation of transition systems by elementary nets and produced an axiomatic characterisation of the realisable transition systems in terms of regions [10,11]. A *region* is a set of states which is uniformly entered or exited by all transitions with an identical label. Two axioms characterize the net realisable transition systems. The State Separation axiom requires that two distinct states are distinguished by at least one region. The Forward Closure axiom requires that for each action (i.e., transition label), a state at which the considered action has no occurrence is separated by a common region from all states where this action occurs. The realisation of transition systems by pure bounded PT-nets, or by general PT-nets, was characterised by similar axioms in [1,2], using extended regions inspired by [4,17]. Moreover, [1] showed that the realisation problem for finite transition systems and PT-nets can be solved in time polynomial in the size of the transition system. Regional axioms were also proposed to characterize transition systems isomorphic to sequential case graphs of elementary nets with inhibitor arcs in [5]. If one considers concurrent behaviours of nets, a step transition system may be realised by a net *iff* it is isomorphic to the step reachability graph of this net. An axiomatic characterisation

of the PT-net realisable step transition systems was given in [17], based on an extension of the idea of regions. It was shown in [2] that this synthesis problem can be solved in time polynomial in the size of the step transition system. A similar characterisation of step transition systems realised by elementary nets with inhibitor arcs was given in [18]. All characterisations recalled above may be retrieved as instances of a general result established in [2] and [3]. Namely, for each class of nets under consideration, the regions of a transition system $\mathcal{T}$ may be identified with the morphisms from $\mathcal{T}$ to a (step) transition system τ typical of this class. An axiomatic characterisation of τ -net realisable transition systems in terms of τ -regions may be established once and for all, using τ as a parameter.

The recent studies [14,15] investigated the realisation of step transition systems by elementary nets with context arcs and with the locally maximal firing rule of [13]. The net realisable step transition systems are characterised by the State Separation axiom, an adapted form of the Forward Closure axiom, and a new Maximality axiom. We will extend this characterisation to arbitrary τ -nets, with step firing policies induced by arbitrary preorders $\preceq$ on steps, using the concept of regions defined as morphisms. For more general step firing policies, we will also show that the net realisable step transition systems may be characterised uniformly by two axioms, the usual State Separation axiom and a more powerful Forward closure axiom, parametric on τ and the step firing policy. All the earlier known results are then instances of the newly obtained characterisation. This study is followed by the definition of an algorithm for the realisation of step transition systems by PT-nets with the policy of maximizing linear rewards of steps, where fixing the reward of elementary transitions is part of the synthesis problem. The simplicity of the algorithm supports our claim that the proposed approach is practical.

2 Background

In this section, we present some basic definitions and facts concerning Petri nets, including a general notion of a type of nets defined by a transition system that captures all possible relationships between places and transitions.

Abelian monoids and transition systems. An *abelian monoid* is a set S with a commutative and associative binary (composition) operation $+$ on S , and a neutral element $\mathbf{0}$. The monoid element resulting from composing n copies of s will be denoted by s^n or $n \cdot s$. In particular, $\mathbf{0} = 0 \cdot s = s^0$. We will refer to some specific abelian monoids: (i) the set of natural numbers $\mathbb{N}$ (including zero) with the arithmetic addition and $\mathbf{0} = 0$; (ii) $\mathbb{S}_{PT} = \mathbb{N} \times \mathbb{N}$ with the pointwise addition operation and $\mathbf{0} = (0, 0)$; and (iii) $\mathbb{S}_{ENC} = \mathbb{N} \times \mathbb{N} \times \{0, 1\} \times \{0, 1\}$ with $\mathbf{0} = (0, 0, 0, 0)$ and the composition operation:

$$(w, x, y, z) + (w', x', y', z') = (w + w', x + x', \min\{1, y + y'\}, \min\{1, z + z'\}) .$$

A *transition system* (Q, S, δ) over an abelian monoid S consists of a set of *states* Q and a partial *transition function* $\delta : Q \times S \rightarrow Q$ such that $\delta(q, \mathbf{0}) = q$ for

every state $q \in Q$. An *initialised* transition system $\mathcal{T} = (Q, S, \delta, q_0)$ is a transition system with an *initial* state $q_0 \in Q$ and such that each state q is *reachable*, i.e., there are $s_1, \dots, s_n$ and $q_1, \dots, q_n = q$ such that $n \geq 0$ and $\delta(q_{i-1}, s_i) = q_i$ for $1 \leq i \leq n$. For every state q of a (non-initialised or initialised) transition system $\mathcal{T}$, we will denote by $\text{enbld}_{\mathcal{T}}(q)$ the set of all monoid elements s which are *enabled* at q , i.e., $\delta(q, s)$ is defined. A transition system is *finite* if it has finitely many states, and the set of enabled elements at any of its states is finite. In the diagrams, an initial state will be represented by a small square and all the remaining nodes by circles. The trivial **0**-labelled transitions will be omitted.

In this paper, transition systems will be used for two rather different purposes. First, initialised transition systems $\mathcal{T}$ over certain free abelian monoids will represent concurrent behaviours of Petri nets. We will call them *step transition systems* or *concurrent reachability graphs*. Second, non-initialised transition systems τ over arbitrary abelian monoids will provide ways to define various classes of nets. We will call them *net types*.

Throughout the paper, we will assume that:

- T is a fixed finite set and $\langle T \rangle$ is the free abelian monoid generated by T , with $\alpha, \beta, \dots$ ranging over its elements.
- $\mathcal{T} = (Q, S, \delta, q_0)$ is a fixed initialised transition system over $S = \langle T \rangle$.
- $\tau = (\mathbb{Q}, \mathbb{S}, \Delta)$ is a fixed non-initialised transition system over an abelian monoid $\mathbb{S}$.

The set T will represent a set of Petri net transitions¹ and so $\langle T \rangle$ — seen as the set of all the multisets² over T — comprises all potential *steps* of all Petri nets over T . And transition systems over $\langle T \rangle$ will be called *step transition systems*.

For all $t \in T$ and $\alpha \in \langle T \rangle$, we will use $\alpha(t)$ to denote the multiplicity of t in α , and so $\alpha = \sum_{t \in T} \alpha(t) \cdot t$. Note that $\langle T \rangle$ is isomorphic to the set of all mappings from T to $\mathbb{N}$ with pointwise addition and the constant map 0 as a neutral element.

PT-nets and net systems. A *PT-net* is a bi-partite graph $N = (P, T, W)$, where P and T are disjoint sets of vertices, respectively called *places* and *transitions*, and $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is a set of directed edges with non-negative integer weights. A *marking* of N is a mapping $M : P \rightarrow \mathbb{N}$, and a *PT-net system* $\mathcal{N} = (P, T, W, M_0)$ is a PT-net with an *initial* marking M_0 . We will use the standard conventions concerning the graphical representation of PT-net systems.

Given a PT-net system $\mathcal{N} = (P, T, W, M_0)$, a step $\alpha \in \langle T \rangle$ is *enabled* and may be *fired* at a marking M if, for every place $p \in P$, $M(p) \geq \sum_{t \in T} \alpha(t) \cdot W(p, t)$. Firing such a step leads to the marking M' , for every $p \in P$ defined by:

$$M'(p) = M(p) + \sum_{t \in T} \alpha(t) \cdot (W(t, p) - W(p, t)) .$$

¹ Transitions in nets differ totally from transitions in transition systems.

² So that, for example, $a^3b = \{a, a, a, b\}$.

We denote this by $M[\alpha]M'$. The *concurrent reachability graph* $CRG(\mathcal{N})$ of $\mathcal{N}$ is the step transition system $CRG(\mathcal{N}) = ([M_0], \langle T \rangle, \delta, M_0)$ where

$$[M_0] = \{M_n \mid \exists \alpha_1, \dots, \alpha_n \exists M_1, \dots, M_{n-1} \forall 1 \leq i \leq n : M_{i-1}[\alpha_i]M_i\}$$

is the set of reachable markings and $\delta(M, \alpha) = M'$ iff $M[\alpha]M'$.

Whenever $M[\alpha]M'$ and α may be decomposed into smaller steps β and γ (i.e., $\alpha(t) = \beta(t) + \gamma(t)$ for all $t \in T$), according to the definition of step semantics, there must exist an *intermediate* marking M'' such that $M[\beta]M''$ and $M''[\gamma]M'$. This specific property is not reflected in our definition of step transition systems which extends Mukund's original definition [17]. In fact, the sole purpose of some step firing policies is to ban certain intermediate markings.

Classes of nets defined by transition systems. $\tau = (\mathbb{Q}, \mathbb{S}, \Delta)$ may be used as a parameter in the definition of a class of nets. Each net-type τ specifies the values (markings) that can be stored within net places ($\mathbb{Q}$), the operations (inscriptions on flow arcs) that a net transition may perform on these values ($\mathbb{S}$), and the enabling condition for steps of transitions (Δ).

Definition 1 (τ -net). A τ -net is a bi-partite graph (P, T, F) , where P and T are respectively disjoint sets of places and transitions, and $F : (P \times T) \rightarrow \mathbb{S}$. A marking of the τ -net is a map $M : P \rightarrow \mathbb{Q}$. A τ -net system $\mathcal{N}$ is a τ -net with an initial marking M_0 .

There is a clear similarity between this definition and the definition of a PT-net system. The only essential difference is that in τ -nets a single $F(p, t)$ can be used to encode *one or two* standard arcs between a place p and transition t . In particular, if in a PT-net system $W(p, t) = m$ and $W(t, p) = n$, then this is now represented by setting $F(p, t) = (m, n) \in \mathbb{S}_{PT}$.

Definition 2 (step semantics). Given a τ -net system $\mathcal{N} = (P, T, F, M_0)$, a step $\alpha \in \langle T \rangle$ is (resource) enabled at a marking M of $\mathcal{N}$ if, for every $p \in P$, $\sum_{t \in T} \alpha(t) \cdot F(p, t) \in \text{enbld}_\tau(M(p))$. We denote this by $\alpha \in \text{enbld}_\mathcal{N}(M)$. The firing of such a step produces the marking M' such that, for every $p \in P$:

$$M'(p) = \Delta\left(M(p), \sum_{t \in T} \alpha(t) \cdot F(p, t)\right)$$

We denote this by $M[\alpha]M'$, and then define the concurrent reachability graph $CRG(\mathcal{N})$ of $\mathcal{N}$ as the step transition system formed by firing inductively from M_0 all possible (resource) enabled steps of $\mathcal{N}$.

As in [3], PT-nets may be retrieved as τ_{PT} -nets where $\tau_{PT} = (\mathbb{N}, \mathbb{S}_{PT}, \Delta_{PT})$ is an infinite transition system over the abelian monoid $\mathbb{S}_{PT}$ such that $\Delta_{PT}(n, (i, o))$ is defined iff $n \geq i$ and is then equal to $n - i + o$ (see Figure 1(a)). Intuitively, $F(p, t) = (i, o)$ means that i is the weight of the arc from p to t , and o the weight of the arc in the opposite direction. To transform a PT-net into a τ_{PT} -net with the same concurrent reachability graph all we need to do is to set $F(p, t) = (W(p, t), W(t, p))$, for all places p and transitions t .

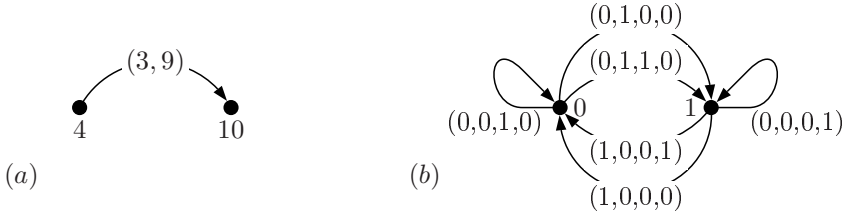


Fig. 1. One of the transitions in τ_{PT} (a); and the finite net type τ_{ENC} (b)

Elementary nets with context arcs under the a-priori step semantics defined in [15] may be retrieved as τ_{ENC} -nets, where τ_{ENC} is a finite transition system over the abelian monoid $\mathbb{S}_{ENC}$ shown in Figure 1(b). Intuitively, if $F(p, t) = (i, o, inh, act)$ then the meaning of i and o are the same as in the case of PT-nets above, while $inh = 1$ indicates that there is an inhibitor arc between p and t , and $act = 1$ indicates that there is an activator (or read) arc between p and t .

Synthesising τ -nets. The synthesis problem is both a *feasibility* problem and an *effective construction* problem. On the one hand, one asks for an exact and hopefully effective (i.e., decidable) characterisation of the transition systems that can be realised by Petri nets. On the other hand, one seeks a constructive procedure for deriving Petri nets from transition systems. Typically, an algorithm constructing such a net is a by-product of an effective solution to the feasibility problem. In the case of τ -nets, the synthesis problem combines the following two sub-problems.

FEASIBILITY

Provide necessary and sufficient conditions for a given $\mathcal{T}$ to be *realised* by some τ -net system $\mathcal{N}$ (i.e., $\mathcal{T} \cong CRG(\mathcal{N})$ where $\cong$ is transition system isomorphism preserving the initial states and transition labels).

EFFECTIVE CONSTRUCTION

Given a finite $\mathcal{T}$, construct a finite τ -net system $\mathcal{N}$ with $\mathcal{T} \cong CRG(\mathcal{N})$.

Note that the decision whether $\mathcal{T}$ can be realised by some τ -net system $\mathcal{N}$ is an implicit sub-problem of the EFFECTIVE CONSTRUCTION problem. Realisable step transition systems have been characterised in [2,3] by two axioms parametric on τ . The axioms require the presence of a sufficiently rich family of τ -regions, defined as morphisms from $\mathcal{T}$ to τ .

Definition 3 (τ -region). A τ -region of $\mathcal{T}$ is a pair (σ, η) , where $\sigma : Q \rightarrow \mathbb{Q}$ and $\eta : \langle T \rangle \rightarrow \mathbb{S}$ is a morphism of monoids, such that for all $q \in Q$ and $\alpha \in \langle T \rangle$:

$$\eta(enbld_{\mathcal{T}}(q)) \subseteq enbld_{\tau}(\sigma(q)) \quad \text{and} \quad \Delta(\sigma(q), \eta(\alpha)) = \sigma(\delta(q, \alpha)) .$$

For every state q of Q , we denote by $enbld_{\mathcal{T}, \tau}(q)$ the set of all steps α such that $\eta(\alpha) \in enbld_{\tau}(\sigma(q))$, for all τ -regions (σ, η) of $\mathcal{T}$.

Note that from the definition of a τ -region it immediately follows that, for every state q of $\mathcal{T}$:

$$\text{enbld}_{\mathcal{T}}(q) \subseteq \text{enbld}_{\mathcal{T},\tau}(q). \quad (1)$$

In the context of the synthesis problem, a τ -region represents a place p whose local state (in τ) is consistent with the global state (in $\mathcal{T}$).

It turns out that $\mathcal{T}$ can be realised by a τ -net system *iff* the following two regional axioms hold:

STATE SEPARATION

For any pair of distinct states q and r of $\mathcal{T}$, there is a τ -region (σ, η) of $\mathcal{T}$ such that $\sigma(q) \neq \sigma(r)$.

FORWARD CLOSURE

For every state q of $\mathcal{T}$, $\text{enbld}_{\mathcal{T},\tau}(q) \subseteq \text{enbld}_{\mathcal{T}}(q)$.

In other words, for every state q of $\mathcal{T}$ and every step α in $\langle T \rangle \setminus \text{enbld}_{\mathcal{T}}(q)$, there is a τ -region (σ, η) of $\mathcal{T}$ such that $\eta(\alpha) \notin \text{enbld}_{\tau}(\sigma(q))$.

In this way, STATE SEPARATION and FORWARD CLOSURE provide a solution to the FEASIBILITY problem. The next issue is to consider the EFFECTIVE CONSTRUCTION problem for a finite $\mathcal{T}$. This can be non-trivial since in general, even though $\mathcal{T}$ may be realized by a τ -net system $\mathcal{N}$, it may be impossible to construct such finite $\mathcal{N}$, as shown by the following example.

Example 1. Let $T = \{a\}$ and $\mathcal{T}$ be a single-node transition system with one 0-loop at the initial state. Moreover, let $\tau = (\mathbb{N} \setminus \{0\}, \{x^n \mid n \geq 0\}, \Delta)$ be a net type such that $\Delta(m, x^n)$ is defined *iff* $n \neq m$. Then there is a τ -net with a concurrent reachability graph isomorphic to $\mathcal{T}$. All we need to take is: $P = \mathbb{N} \setminus \{0\}$, $T = \{a\}$, and for all $n \geq 1$, $F(n, a) = x$ and $M_0(n) = n$. However, no finite τ -net can generate $\mathcal{T}$ as one place can disable at most one step a^n . $\square$

A solution to the EFFECTIVE CONSTRUCTION problem is obtained if one can compute a finite set $\mathcal{WR}$ of τ -regions of $\mathcal{T}$ *witnessing* the satisfaction of all instances of the regional axioms [9]. A suitable τ -net system is then $\mathcal{N}_{\mathcal{WR}} = (P, T, F, M_0)$ where $P = \mathcal{WR}$ and, for any place $p = (\sigma, \eta)$ in P , $F(p, t) = \eta(t)$ and $M_0(p) = \sigma(q_0)$ (recall that q_0 is the initial state of $\mathcal{T}$, and $T \subseteq \langle T \rangle$).

In the case of PT-nets, τ_{PT} -regions coincide with the regions defined in [17] where a similar characterisation of PT-net realisable step transition systems is established. The EFFECTIVE CONSTRUCTION problem may be solved [2] using time polynomial in $|Q|$, $|T|$, the maximal size of a step α such that $\delta(q, \alpha)$ is defined for an arbitrary q , and the maximal number of minimal steps α such that $\delta(q, \alpha)$ is undefined for fixed q .

In the case of elementary nets with context arcs, τ_{ENC} -regions coincide with the regions defined in [15], and the EFFECTIVE CONSTRUCTION problem can be solved because the number of τ_{ENC} -regions of a finite $\mathcal{T}$ is also finite (but this problem is NP-complete).

Note that any axiomatic solution to the FEASIBILITY problem should entail the equality $\text{enbld}_{\mathcal{T}}(q) = \text{enbld}_{\mathcal{T},\tau}(q)$. In view of the trivially holding inclusion (1), FORWARD CLOSURE is quite sufficient for this purpose. However, in the

context of firing policies, no such ‘trivial’ inclusion holds, and the FORWARD CLOSURE axiom will be in terms of equality rather than inclusion between sets of steps.

3 Step Firing Policies

We introduce formally step firing policies after several motivating examples. Most examples show simple step firing policies induced by a pre-order $\preceq$ on the steps in $\langle T \rangle$, with the outcome that a resource enabled step α is (control) disabled whenever there is another resource enabled step β such that $\alpha \prec \beta$ (i.e., $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$). However, the first example below demands more.

Example 2. In the context of an electronic system design, suppose that a and b are transitions denoting requests served by an arbiter which maintains mutual exclusion on a common non-shareable resource, and m is a transition representing some testing action on this arbiter, e.g., its internal state sampling. The condition for activating m is such that the testing procedure can only happen if the arbiter logic is in a well-defined binary state. That is, we cannot reliably test the arbiter if it is in the *conflict resolution* state, where *both* requests are present (although only one can be served). In such a conflict state the arbiter may experience a metastable or oscillatory condition, and therefore it cannot be tested reliably.

The above condition leads to the policy that, in a net model of the system, transition m should not be allowed to fire in any marking where the requesting transitions, a and b , are both enabled. Otherwise, m should be allowed. This behaviour can be implemented using a safe (or 1-bounded) PT-net system together with a policy that prevents m from being fired whenever both a and b are also enabled. Without such a policy, no safe PT-net system can realise this behaviour, though there is a 2-bounded net realisation as well as a safe net realisation with label splitting, constructed by the Petrify tool [7]. $\square$

The above example also shows that firing policies can sometimes be avoided by resorting to more general types of nets, i.e., by changing the trade-off between resource enabledness and control enabledness. However, as the next example shows, allowing non-safe markings may not be enough for this purpose.

Example 3. Consider the step transition system in Figure 2(a). It is easy to see that it cannot be realised by any PT-net nor by any standard Petri net executed under the usual resource based enabling since, for example, it does not allow the step a in the initial state. However, if we execute the PT-net in Figure 2(b) under the maximal concurrency rule, then what we get is exactly the target step transition system. (Note that maximal concurrency can be captured by the preorder on $\langle T \rangle$ corresponding to multiset inclusion.) $\square$

Example 4. Elementary nets in [14] and elementary nets with context arcs in [15] have been supplied with a concurrency semantics more restrictive than Definition 2. In both papers, the set of transitions T is partitioned by a *location* equivalence $\equiv$, and the net is executed by firing the maximal enabled steps w.r.t. the

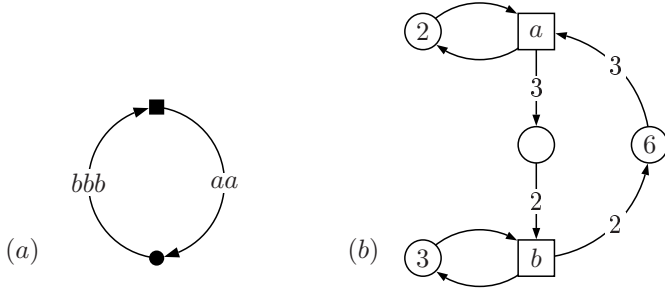


Fig. 2. Transition system which cannot be realised without a step firing policy (a) and the underlying PT-net for its realisation (b)

following preorder (steps here are simply sets, or maps $T \rightarrow \{0, 1\}$): $\alpha \preceq \beta$ iff for all $t \in T$, $\alpha(t) = 1 \Rightarrow \beta(t) = 1$ and $\beta(t) = 1 \Rightarrow (\exists t') (t \equiv t' \wedge \alpha(t') = 1)$. $\square$

Example 5. To capture the behaviour of PT-nets with static priorities and the interleaving semantics, it suffices to state that $t \preceq t'$ if the priority of t is lower than or equal to the priority of t' , and that $t + \alpha \prec t$ for all $\alpha \neq \mathbf{0}$. $\square$

Example 6. To maximize an objective function one can take a *reward* map $\$: T \rightarrow \mathbb{Z}$ and assume that: $\alpha \preceq_{\$} \beta$ iff $\sum_{t \in T} \alpha(t) \cdot \$(t) \leq \sum_{t \in T} \beta(t) \cdot \$(t)$. Figure 3 shows an interesting consequence of applying the reward based step firing policy with $\$(a) = 1$ and $\$(b) = -1$ to the PT-net in Figure 3(a). The resulting concurrent reachability graph, shown in Figure 3(b), has strictly sequential behaviour, yet in the purely resource based view the transitions a and b are concurrent. We also show the resource based behaviour in Figure 3(c) to illustrate the degree to which applying policies can trim down the original executions. $\square$

Example 7. Consider a PT-net consisting of one transition a and no places, executed under a policy induced by the preorder such that $a^m \preceq a^n$ iff $m \leq n$. In such a net, all steps over $\langle T \rangle$ are resource enabled, yet they are all disabled by the firing control policy. $\square$

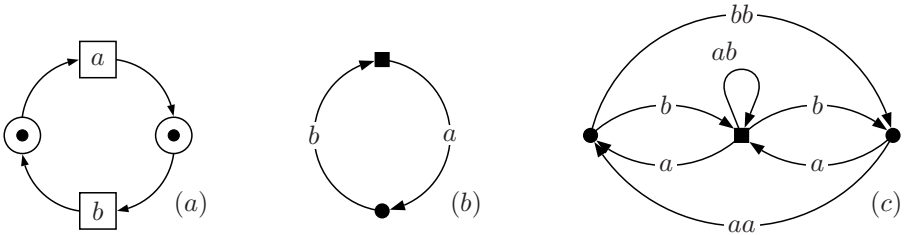


Fig. 3. The effect of applying a reward based firing control policy

To conclude the above discussion, we feel that step firing policies are a way of capturing synchronous aspects of concurrent (or asynchronous) systems. Moreover, if one wants to realise a concurrent transition system, the right discipline is to choose first a closely matching and suitable *net-type*, and then to use a *step firing policy* only as a last resort for the fine tuning of the behaviour.

We will now properly define step firing policies and their effect on net behaviour. Throughout the rest of the section, $\mathcal{X}_\tau$ is a family comprising all sets $enbld_{\mathcal{N}}(M)$ such that $\mathcal{N}$ is a τ -net system and M is a reachable marking of $\mathcal{N}$. In other words, $\mathcal{X}_\tau$ contains any set of enabled steps in any reachable marking of any τ -net with set of transitions T . In a sense, this is all we need to care about for defining and enforcing control policies on τ -net systems.

Definition 4 (step firing policy). *A step firing policy over $\langle T \rangle$ is given by a control disabled steps mapping $cds : 2^{\langle T \rangle} \rightarrow 2^{\langle T \rangle}$ such that, for all $X \subseteq \langle T \rangle$ and $Y \subseteq X$:*

1. $cds(X) \subseteq X$;
2. $cds(Y) \subseteq cds(X)$; and
3. $X \in \mathcal{X}_\tau$ and $X \setminus cds(X) \subseteq Y$ imply $cds(X) \cap Y \subseteq cds(Y)$.

In the above, X should be understood as the set of resource enabled steps at some reachable marking of some τ -net system, and $X \setminus cds(X)$ as the subset of *control enabled* steps at this marking. The condition $cds(X) \subseteq X$ has been included only for intuitive reasons and could be dropped. To see the rationale behind the other conditions, imagine that control disabling a step $\alpha \in Y$ is due to the (resource) enabling of some set of steps $DisSet_1^\alpha$ or $DisSet_2^\alpha$ or $\dots$. Then Definition 4(2) simply states that each set $DisSet_i^\alpha$ that is present in Y is also present in X and so disables α in X as well. Definition 4(3) is more subtle and conveys the intuition that at least one disabling set $DisSet_k^\alpha$ survives, i.e., $DisSet_k^\alpha \subseteq X \setminus cds(X)$, and is present in Y . So, if $\alpha \in Y$ and was control disabled in X then it will also be control disabled in Y . Hence the last two conditions capture different types of monotonicity.

Definition 5 (applying step firing policy). *Let $\mathcal{N}$ be a τ -net system, and cds be a step firing policy for $\mathcal{N}$. Applying cds to $\mathcal{N}$ consists in control disabling at each marking M all the resource enabled steps that belong to $cds(enbld_{\mathcal{N}}(M))$. We will use $CRG_{cds}(\mathcal{N})$ to denote the induced reachable restriction of $CRG(\mathcal{N})$. (Note that $CRG_{cds}(\mathcal{N})$ may be finite even though $CRG(\mathcal{N})$ is infinite.) Moreover,*

$$enbld_{\mathcal{N}, cds}(M) = enbld_{\mathcal{N}}(M) \setminus cds(enbld_{\mathcal{N}}(M))$$

will denote the set of steps enabled at reachable marking M under policy cds .

One salient feature of the classical net theory is that adding a place to a Petri net can only reduce its behaviour and, in particular, can never turn a disabled step into an enabled one. It is important to observe that when considering step firing policies, this is no more the case : *adding places can add new behaviours*.

For example, adding a place that disables a transition with high priority may result in control enabling a transition with lower priority.

Example 7 shows a situation where there is an infinite chain of steps each one control disabled by its successor. This dubious situation which raises implementation problems is ruled out by condition (3) in Definition 4(3). In order that a preorder $\preceq$ on $\langle T \rangle$ induces a step firing policy that can reasonably be applied to a τ -net $\mathcal{N}$, the considered preorder should be *locally noetherian* w.r.t. $\mathcal{N}$, which means that no infinite chain of steps $\alpha_1 \prec \dots \prec \alpha_i \prec \dots$ is ever (resource) enabled at any marking in $CRG(\mathcal{N})$.

The step firing policy induced by a preorder relation $\preceq$ on $\langle T \rangle$ is given by:

$$cds_{\preceq}(X) = \{\alpha \in X \mid (\exists \beta \in X) \alpha \prec \beta\}.$$

Such a policy consists in control disabling all those resource enabled steps that fail to be $\preceq$ -maximal.

Proposition 1. *If a preorder $\preceq$ on $\langle T \rangle$ is locally noetherian w.r.t. every τ -net system, then $cds_{\preceq}$ is a step firing policy.*

Proof. Definition 4(1) is clearly satisfied. To show Definition 4(2), we take $Y \subseteq X$ and $\alpha \in cds_{\preceq}(Y)$. Then, by the definition of $cds_{\preceq}$, there is $\beta \in Y \subseteq X$ such that $\alpha \prec \beta$. Hence, by the definition of $cds_{\preceq}$, $\alpha \in cds_{\preceq}(X)$.

To show Definition 4(3), assume that $X \in \mathcal{X}_{\tau}$, $Y \subseteq X$, $X \setminus cds_{\preceq}(X) \subseteq Y$ and $\alpha \in cds_{\preceq}(X) \cap Y$. We need to show that $\alpha \in cds_{\preceq}(Y)$.

From $\alpha \in cds_{\preceq}(X)$ it follows that there is $\beta \in X$ such that $\alpha \prec \beta$. We now consider two cases.

Case 1: $\beta \in Y$. Then, by the definition of $cds_{\preceq}$, $\alpha \in cds_{\preceq}(Y)$.

Case 2: $\beta \in X \setminus Y$. Then, by $X \setminus cds_{\preceq}(X) \subseteq Y$, we have that $\beta \in cds_{\preceq}(X)$. Hence, there is $\gamma \in X$ such that $\beta \prec \gamma$. If $\gamma \in Y$ we have Case 1 with $\beta = \gamma$ and obtain $\alpha \in cds_{\preceq}(Y)$ due to the transitivity of $\prec$. Otherwise, we have Case 2 with $\beta = \gamma$ and so $\gamma \in cds_{\preceq}(X)$. And then we repeat the same argument. Now, because $\preceq$ is locally noetherian w.r.t. all τ -net systems and $X \in \mathcal{X}_{\tau}$, one must find sooner or later in this iteration some $\phi \in Y$ such that Case 1 holds with $\beta = \phi$, and so $\alpha \in cds_{\preceq}(Y)$. $\square$

Not all practically relevant step firing policies can be defined by preorders. For instance, the one described in Example 2 cannot be dealt with by any $cds_{\preceq}$ (note that the transitions a, b, m never occur all in the same step as the resource cannot be shared). However, this example is easily rendered with the policy cds such that $\mu \in cds(X)$ iff $\mu(m) \geq 1$ and there are $\alpha, \beta \in X$ such that $\alpha(a), \beta(b) \geq 1$.

We would like to add that Definition 4 came after the earlier and more intuitive definition of step firing policies based on preorders. Net synthesis was studied first for these preorder based policies. Definition 4 was found while trying to simplify the technical development in this earlier attempt. Definition 4 supersedes the earlier definition, has a built-in noetherian condition (3), and it allowed us to give a more compact characterization of the net realisable transition systems (without the *maximality* axiom).

4 Characterisation of Net Realisable Transition Systems

We now enter the second, more technically oriented, part of this paper where we will re-visit the net synthesis problem but this time taking into account step firing policies. Throughout this section, we assume that cds is a fixed step firing policy over $\langle T \rangle$.

FEASABILITY WITH POLICIES

Provide necessary and sufficient conditions for a given $\mathcal{T}$ to be *realised* by some τ -net system $\mathcal{N}$ executed under the given step firing policy cds over $\langle T \rangle$, i.e., $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$.

In this characterisation, we do not require that $\mathcal{N}$ is finite. If such a finite τ -net system does exist, then we call $\mathcal{T}$ *finitely realisable*.

We first provide an auxiliary result showing that, when a τ -net realises a transition system, every place of the τ -net determines a corresponding τ -region of the transition system.

Proposition 2. *Let $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$ for a τ -net system $\mathcal{N} = (P, T, F, M_0)$. Then, for each place $p \in P$, there is exactly one τ -region (σ, η) of $\mathcal{T}$ such that $\sigma(q_0) = M_0(p)$ and, for all $q \in Q$ and $\alpha \in \text{enbld}_{\mathcal{T}}(q)$:*

$$\eta(\alpha) = \sum_{t \in T} \alpha(t) \cdot F(p, t) .$$

Proof. All step transition systems under consideration are deterministic (because we use functions rather than relations to capture transitions in step transition systems), and $\mathcal{T}$ is reachable. Therefore, $\sigma(q_0)$ and $\eta : \langle T \rangle \rightarrow \mathbb{S}$ determine at most one map $\sigma : Q \rightarrow \mathbb{Q}$ such that $\Delta(\sigma(q), \eta(\alpha)) = \sigma(\delta(q, \alpha))$ whenever $\alpha \in \text{enbld}_{\mathcal{T}}(q)$, and hence they determine at most one τ -region of $\mathcal{T}$. It remains to exhibit such a map σ .

Now $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$ (by assumption) and $CRG_{cds}(\mathcal{N}) \hookrightarrow CRG(\mathcal{N})$ (by definition) where the embedding is the inclusion of states and transitions. Let $\sigma : Q \rightarrow \mathbb{Q}$ be the map defined by $\sigma(q) = f(q)(p)$, where $f(q)$ is the image of q through the isomorphism $\cong$ (thus $f(q)$ is a marking of $\mathcal{N}$). Let $\eta : \langle T \rangle \rightarrow \mathbb{S}$ be as stated in the proposition. In view of Definition 2, σ satisfies $\eta(\alpha) \in \text{enbld}_{\tau}(\sigma(q))$ and $\Delta(\sigma(q), \eta(\alpha)) = \sigma(\delta(q, \alpha))$ whenever $\alpha \in \text{enbld}_{\mathcal{T}}(q)$, and it also satisfies $\sigma(q_0) = M_0(p)$ because $\cong$ is an isomorphism. The result therefore holds. $\square$

We now can present the central result of the paper, which is based on the following modification of the FORWARD CLOSURE axiom:

FORWARD CLOSURE WITH POLICIES

For every state q of $\mathcal{T}$, $\text{enbld}_{\mathcal{T}}(q) = \text{enbld}_{\mathcal{T}, \tau}(q) \setminus cds(\text{enbld}_{\mathcal{T}, \tau}(q))$.

Theorem 1 ([8]). $\mathcal{T}$ is realisable ($\mathcal{T} \cong CRG_{cds}(\mathcal{N})$ for some $\mathcal{N}$) iff $\mathcal{T}$ satisfies STATE SEPARATION and FORWARD CLOSURE WITH POLICIES.

In many contexts of practical application, the general theory of step firing policies may be specialised to firing policies induced by preorders on steps. In this particular case, the FEASIBILITY problem may be restated as follows.

FEASIBILITY WITH PREORDERS

Given a preorder $\preceq$ on steps, provide necessary and sufficient conditions for a given $\mathcal{T}$ to be realised by some τ -net system $\mathcal{N}$ such that $\preceq$ is locally noetherian w.r.t. $\mathcal{N}$ and $\mathcal{N}$ is executed under the step firing policy induced by $\preceq$.

The following three axioms, in addition to STATE SEPARATION, provide a solution to the above problem:

MAXIMALITY

For every state q of $\mathcal{T}$ and all α, β in $enbld_{\mathcal{T}}(q)$, if $\alpha \preceq \beta$ then $\beta \preceq \alpha$.

LOCAL BOUNDEDNESS

For every state q of $\mathcal{T}$ and every infinite chain of steps $\alpha_1 \prec \dots \prec \alpha_i \prec \dots$ in $\langle T \rangle$, there is a τ -region (σ, η) of $\mathcal{T}$ and $i \geq 1$ such that $\eta(\alpha_i) \notin enbld_{\tau}(\sigma(q))$.

FORWARD CLOSURE WITH PREORDERS

For every state q of $\mathcal{T}$ and every step α in $\langle T \rangle \setminus enbld_{\mathcal{T}}(q)$, there is a τ -region (σ, η) of $\mathcal{T}$ such that $\eta(\alpha) \notin enbld_{\tau}(\sigma(q))$, or there is $\beta \in enbld_{\mathcal{T}}(q)$ such that $\alpha \prec \beta$.

Let us make some remarks upon LOCAL BOUNDEDNESS. For many classes of nets, including, e.g., the elementary nets, this axiom is superfluous since any preorder on $\langle T \rangle$ is locally noetherian w.r.t. any τ -net system. The role of LOCAL BOUNDEDNESS is to preclude the resource enabling of an infinite chain of steps $\alpha_1 \prec \dots \prec \alpha_i \prec \dots$ at some reachable marking of the synthesised net, because this would result in control disabling all of them. Step firing policies would better avoid this paradox!

5 Construction of Nets Realising Transition Systems

We finally consider the constructive part of the synthesis problem for τ -nets with step firing policies:

EFFECTIVE CONSTRUCTION WITH FIXED POLICIES

Given a finite $\mathcal{T}$ and a step firing policy cds over $\langle T \rangle$, construct a finite τ -net system $\mathcal{N}$ such that $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$.

For a finite net-type τ , one can always proceed by exhaustive enumeration of τ -regions, but this is very inefficient. For elementary nets with localities, an algorithmic approach based on the axiomatic characterisation of Theorem 1 has been initiated, but still some work needs to be done to improve the efficiency. For

PT-nets with the maximal step firing policy (induced from multiset inclusion), it suffices to check $\mathcal{T}$ for the *maximality* axiom and then to apply the synthesis algorithm defined in [2] (for unconstrained step firing).

Now, instead of imposing a fixed firing policy cds , one may prefer to specify a *family* $\mathcal{CDS}$ of step firing policies in view of solving the following problem:

EFFECTIVE CONSTRUCTION WITH UNKNOWN POLICIES

Given a finite $\mathcal{T}$ and a family $\mathcal{CDS}$ of step firing policies over $\langle T \rangle$, construct a finite τ -net system $\mathcal{N}$ and select cds in $\mathcal{CDS}$ such that $\mathcal{T} \cong CRG_{cgs}(\mathcal{N})$.

Again, for finite net-types τ and finite families of step firing policies $\mathcal{CDS}$, one can always proceed by exhaustive enumeration, but one can sometimes do much better. In order to support this claim, we propose in the rest of this section a decision and synthesis algorithm for PT-nets with step firing policies aiming at maximizing linear rewards of steps, where fixing the reward of the elementary transitions is part of the synthesis problem.

Synthesising PT-nets with linear rewards of steps. As already argued, PT-nets may be seen as τ_{PT} -nets, and the concurrent reachability graph $CRG(\mathcal{N})$ of a PT-net system $\mathcal{N}$ coincides then with $CRG_{cgs}(\mathcal{N})$ where cgs is the step firing policy induced from the equality of steps. Our goal is to provide a solution to the following specific instance of the net synthesis problem:

FEASABILITY WITH UNKNOWN REWARDS

Provide necessary and sufficient conditions for a given $\mathcal{T}$ to be realised by some PT -net system $\mathcal{N}$ under the step firing policy induced by some reward map $\$: T \rightarrow \mathbb{Z}$ as defined in Example 6, i.e., $\mathcal{T} \cong CRG_{cgs}(\mathcal{N})$.

EFFECTIVE CONSTRUCTION WITH UNKNOWN REWARDS

Given a finite $\mathcal{T}$, construct a finite PT -net system $\mathcal{N}$ and a reward map $\$: T \rightarrow \mathbb{Z}$ such that $\mathcal{T} \cong CRG_{cgs}(\mathcal{N})$.

However, the statement of the above problems is not totally correct, for the preorder relations defined from reward maps are generally not locally noetherian for all PT-nets. This difficulty may be smoothed away by adapting Definition 4, Proposition 1, and Theorem 1 as follows.

Definition 6 (bounded step firing policy). A bounded step firing policy for τ -nets over $\langle T \rangle$ is a mapping $cgs : 2^{\langle T \rangle} \rightarrow 2^{\langle T \rangle}$ such that, for all $X \subseteq \langle T \rangle$ and $Y \subseteq X$:

1. $cgs(X) \subseteq X$;
2. X infinite $\Rightarrow cgs(X) = \emptyset$
3. X finite $\Rightarrow cgs(Y) \subseteq cgs(X)$,
4. $X \in \mathcal{X}_\tau$ and $X \setminus cgs(X) \subseteq Y$ imply $cgs(X) \cap Y \subseteq cgs(Y)$.

Definition 7. Given a preorder relation $\preceq$ on $\langle T \rangle$, let $cgs_{\preceq}^b : 2^{\langle T \rangle} \rightarrow 2^{\langle T \rangle}$ be the map such that $cgs_{\preceq}^b(X) = \emptyset$ if X is infinite and otherwise $cgs_{\preceq}^b(X) = \{\alpha \in X \mid (\exists \beta \in X) \alpha \prec \beta\}$.

Proposition 3. *If $\preceq$ is a preorder on $\langle T \rangle$, then $cds_{\preceq}^b$ is a bounded step firing policy over $\langle T \rangle$.*

Theorem 2. *Let $\mathcal{T}$ be finite and cds be a bounded step firing policy for $\langle T \rangle$. Then $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$ for some τ -net system $\mathcal{N}$ iff $\mathcal{T}$ satisfies STATE SEPARATION and FORWARD CLOSURE WITH POLICIES.*

The exact problem we want to solve is the following: given a finite $\mathcal{T}$, decide whether there exists and construct a finite PT -net system $\mathcal{N}$ and a reward map $\$: T \rightarrow \mathbb{Z}$ such that $\mathcal{T} \cong CRG_{cds}(\mathcal{N})$ for $cds = cds_{\preceq}^b$. In order to derive a decision and synthesis procedure from the axiomatic characterisation provided by Theorem 2, one needs to compute an effective representation of the set of all τ_{PT} -regions of $\mathcal{T}$. This can be done as follows.

First, one constructs a spanning tree for $\mathcal{T} = (Q, S, \delta, q_0)$, i.e., a reachable step transition system $Tr = (Q, S, \delta', q_0)$ such that, for all $q \in Q$ and $\alpha \in \langle T \rangle$, $\delta'(q, \alpha) = \delta(q, \alpha)$ or it is undefined, and $\delta'(q, \alpha) = \delta'(q', \alpha')$ entails $q = q'$ and $\alpha = \alpha'$. The labelled arcs $q \xrightarrow{\alpha} q'$ in $\mathcal{T} \setminus Tr$ (such that $\delta(q, \alpha) = q'$ and $\delta'(q, \alpha)$ is undefined) are called chords. Each chord $q \xrightarrow{\alpha} q'$ determines a basic cycle in $\mathcal{T}$ as follows. Let q'' be the nearest common ancestor of q and q' in the spanning tree Tr , then the path from q'' to q , the chord $q \xrightarrow{\alpha} q'$, and the reverse path from q' to q'' form a cycle in $\mathcal{T}$. The label of this cycle is a sequence $sign_1(\alpha_1) \dots sign_n(\alpha_n)$ where the α_i are steps and the signs (+ or -) distinguish between the forward and the reverse arcs.

From Definition 3 and by observing that $\mathcal{T}$ is reachable, a τ -region (σ, η) of $\mathcal{T}$ is totally determined by $\sigma(q_0)$ and the map η , which is determined in turn by the values $\eta(t)$ for all $t \in T$. In the case of τ_{PT} -regions, $\sigma(q_0) \in \mathbb{N}$ and $\eta(t) \in \mathbb{N} \times \mathbb{N}$ for all t . Suppose that we have variables p^{init} and $t \bullet p$ and $p \bullet t$ for all $t \in T$. A τ_{PT} -region (σ, η) of $\mathcal{T}$ may be represented as a non-negative integer valuation of these variables, viz. $\sigma(q_0) = p^{init}$ and $\eta(t) = (p \bullet t, t \bullet p)$ for all t . Conversely, in order that a non-negative integer valuation of the variables p^{init} , $t \bullet p$ and $p \bullet t$ would define a τ_{PT} -region of $\mathcal{T}$, it is *necessary and sufficient* that: (i) for each basic cycle $sign_1(\alpha_1) \dots sign_n(\alpha_n)$ we have:

$$\sum_i \sum_{t \in T} (sign_i(\alpha_i(t))) \cdot (t \bullet p - p \bullet t) = 0 ;$$

and (ii) for each path from q_0 to q labelled with $\alpha_1 \dots \alpha_n$ in the spanning tree, and for each β such that $\delta(q, \beta)$ is defined in $\mathcal{T}$ we have:

$$p^{init} + \sum_i \sum_{t \in T} \alpha_i(t) \cdot (t \bullet p - p \bullet t) \geq \sum_{t \in T} \beta(t) \cdot p \bullet t .$$

Let $\mathcal{R}$ denote the finite system formed from these linear constraints. The inequalities in $\mathcal{R}$ guarantee that $\eta(enbld_{\mathcal{T}}(q)) \subseteq enbld_{\tau}(\sigma(q))$ assuming that

$$\sigma(q) = p^{init} + \sum_i \sum_{t \in T} \alpha_i(t) \cdot (t \bullet p - p \bullet t) .$$

The equations in $\mathcal{R}$ guarantee that the above definition of σ is compatible with the implicit constraints induced from the chords $q \xrightarrow{\alpha} q'$ (i.e., $\delta(q, \alpha) = q'$):

$$\sigma(q') = \sigma(q) + \sum_{t \in T} \alpha(t) \cdot (t^\bullet p - p^\bullet t).$$

Now checking STATE SEPARATION for two distinct states q and r reached from q_0 by paths with respective labels $\alpha_1 \dots \alpha_n$ and $\beta_1 \dots \beta_m$ in the spanning tree Tr amounts to deciding whether there exists a place (region) p such that

$$\sum_i \sum_{t \in T} \alpha_i(t) \cdot (t^\bullet p - p^\bullet t) \neq \sum_j \sum_{t \in T} \beta_j(t) \cdot (t^\bullet p - p^\bullet t).$$

This can be decided within time polynomial in the size of $\mathcal{R}$.

Checking FORWARD CLOSURE WITH POLICIES is somewhat more complicated since on the one hand, the condition $enbld_{\mathcal{T}}(q) = enbld_{\mathcal{T}, \tau}(q) \setminus cds(enbld_{\mathcal{T}, \tau}(q))$ bears upon the *infinite* set of τ_{PT} -regions of $\mathcal{T}$, and on the other hand, $cds = cds_{\leq_s}^b$ depends on an unknown map $\$: T \rightarrow \mathbb{Z}$. Fortunately, all sets $enbld_{\mathcal{T}, \tau}(q)$ are finite, hence the constraint that $cds_{\leq_s}^b(X) = \emptyset$ for any infinite set X is not a problem, and they may be computed from a *finite* set R of τ_{PT} -regions of $\mathcal{T}$.

The first claim may be justified easily. For each $t \in T$, let $degree(t)$ be the maximum of $\alpha(t)$ for all steps α such that $\delta(q, \alpha)$ is defined for some $q \in Q$ (such a maximum exists since $\mathcal{T}$ is finite). Then, for each t , $p^{init} = degree(t)$, $t^\bullet p = p^\bullet t = 1$, and $t'^\bullet p = p^\bullet t' = 0$ for $t' \neq t$ defines a τ_{PT} -region of $\mathcal{T}$. Therefore, no step α with $\alpha(t) > degree(t)$ belongs to $enbld_{\mathcal{T}, \tau}(q)$ for any $q \in Q$.

We establish now the second claim. As all equations and inequalities in $\mathcal{R}$ are linear and homogeneous (their constant term is always 0), the set of rational solutions of this finite system forms a polyhedral cone in $\mathbb{Q}^{2|T|+1}$. Any polyhedral cone has a finite set of generating rays (see [19]), and this set can be computed [6]. Let $\{p_j \mid j = 1 \dots m\}$ be the set of generating rays of the cone defined by $\mathcal{R}$, where each ray p_j is given by a non-negative integer value p_j^{init} and two non-negative integer maps $p_j^\bullet, \bullet p_j : T \rightarrow \mathbb{N}$. Each ray p_j represents a corresponding τ_{PT} -region (σ_j, η_j) of $\mathcal{T}$. Since the p_j generate all solutions of $\mathcal{R}$, any other τ_{PT} -region of $\mathcal{T}$ is a linear combination $(\sigma, \eta) = \sum_{j=1}^m r_j \cdot (\sigma_j, \eta_j)$ of generating regions with non-negative rational coefficients r_j .

Let $q \in Q$ and let $\alpha_1 \dots \alpha_n$ label the path from q_0 to q in the spanning tree. A step α does not belong to $enbld_{\mathcal{T}, \tau}(q)$ iff

$$p^{init} + \sum_i \sum_{t \in T} \alpha_i(t) \cdot (t^\bullet p - p^\bullet t) < \sum_{t \in T} \alpha(t) \cdot p^\bullet t$$

for some (non-negative) integer solution of the linear system $\mathcal{R}$. Suppose this inequality holds for

$$p^{init} = \sum_{j=1}^m r_j \cdot p_j^{init} \quad \text{and} \quad p^\bullet t = \sum_{j=1}^m r_j \cdot p_j^\bullet t \quad \text{and} \quad t^\bullet p = \sum_{j=1}^m r_j \cdot t^\bullet p_j$$

where the r_j are non-negative rational coefficients. Then necessarily,

$$p_j^{init} + \sum_i \sum_{t \in T} \alpha_i(t) \cdot (t \bullet p_j - p_j \bullet t) < \sum_{t \in T} \alpha(t) \cdot p_j \bullet t$$

for some j . Therefore, all finite sets of enabled steps $enbld_{\mathcal{T},\tau}(q)$ can be computed from the finite set of τ_{PT} -regions $R = \{(\sigma_j, \eta_j) \mid 1 \leq j \leq m\}$.

The last problem we are facing is to check whether $enbld_{\mathcal{T}}(q) = enbld_{\mathcal{T},\tau}(q) \setminus cds(enbld_{\mathcal{T},\tau}(q))$ for all q and for some bounded step firing policy $cds_{\leq \mathbb{S}}^b$, induced by a reward map $\mathbb{S} : T \rightarrow \mathbb{Z}$ yet to be determined. Recall that, for finite X , $cds_{\leq}^b(X) = \{\alpha \in X \mid (\exists \beta \in X) \alpha \prec \beta\}$, $enbld_{\mathcal{T}}(q) \subseteq enbld_{\mathcal{T},\tau}(q)$, and $enbld_{\mathcal{T},\tau}(q)$ is finite for all q . Checking FORWARD CLOSURE WITH POLICIES amounts therefore to deciding upon the existence of a map $\mathbb{S} : T \rightarrow \mathbb{Z}$ such that for all $q \in Q$, $\alpha, \beta \in enbld_{\mathcal{T}}(q)$ and $\gamma \in enbld_{\mathcal{T},\tau}(q) \setminus enbld_{\mathcal{T}}(q)$:

$$\sum_{t \in T} \alpha(t) \cdot \mathbb{S}(t) = \sum_{t \in T} \beta(t) \cdot \mathbb{S}(t) \quad \text{and} \quad \sum_{t \in T} \gamma(t) \cdot \mathbb{S}(t) < \sum_{t \in T} \beta(t) \cdot \mathbb{S}(t).$$

Since all sets $enbld_{\mathcal{T},\tau}(q) \setminus enbld_{\mathcal{T}}(q)$ are finite and effectively computable, the set of all such linear homogeneous equations and inequalities in the variables $\mathbb{S}(t)$ is finite and one can decide whether there exists and compute a solution map $\mathbb{S} : T \rightarrow \mathbb{Z}$. Altogether, we have established the following result.

Theorem 3. *The problem of finding out whether for a given finite $\mathcal{T}$ there is a finite PT-net system $\mathcal{N}$ and a map $\mathbb{S} : T \rightarrow \mathbb{Z}$ such that $\mathcal{T} \cong CRG_{cds_{\leq \mathbb{S}}^b}(\mathcal{N})$ is decidable.*

Moreover, when the problem is feasible, one can extract from the decision procedure a finite PT-net $\mathcal{N} = (P, T, F, M_0)$ such that $\mathcal{T} \cong CRG_{cds_{\leq \mathbb{S}}^b}(\mathcal{N})$ where $\mathbb{S} : T \rightarrow \mathbb{Z}$ is the solution map computed in the above. It suffices actually to let $P = \{p_1, \dots, p_m\}$ be the set of generating rays of the cone defined by $\mathcal{R}$, and to set $M_0(p_j) = p_j^{init}$, and $F(p_j, t) = (p_j \bullet t, t \bullet p_j)$ for all t . The set of places P of $\mathcal{N}$ represents the set of generating τ_{PT} -regions $R = \{(\sigma_j, \eta_j) \mid 1 \leq j \leq m\}$. It is easily seen that two states q and r of $\mathcal{T}$ are separated by some τ_{PT} -region of $\mathcal{T}$ iff they are separated by some region in R . Therefore, as $\mathcal{T}$ satisfies STATE SEPARATION, the map ϕ that sends a state q to the marking defined by $M(p_j) = \sigma_j(q)$ ($1 \leq j \leq m$) embeds $\mathcal{T}$ into $CRG(\mathcal{N})$ in such a way that $enbld_{\mathcal{T}}(q) \subseteq enbld_{\mathcal{T},\tau}(q) = enbld_{CRG(\mathcal{N})}(\phi(q))$, for all q . As $\mathcal{T}$ satisfies FORWARD CLOSURE WITH POLICIES, it follows from Definition 5 that $enbld_{CRG_{cds_{\leq \mathbb{S}}^b}(\mathcal{N})}(\phi(q)) = enbld_{\mathcal{T}}(q)$, for all q . Hence $\mathcal{T} \cong CRG_{cds_{\leq \mathbb{S}}^b}(\mathcal{N})$, as required. Summing up, one can state the following result.

Theorem 4. *The EFFECTIVE CONSTRUCTION problem has a solution for PT-nets WITH UNKNOWN REWARDS OF steps.*

6 Future Work

By posing and providing answers to some basic questions concerning Petri nets with step firing policies, this paper has opened up a new, potentially fertile

research direction of practical importance. Among immediate issues which we have not yet addressed here, or just touched upon, are:

- A characterisation of the interplay between net types and step firing policies.
- A classification of step firing policies according to their practical implementability (one would presumably exclude the policy from Example 7).
- Synthesis from behavioural representations other than finite step transition systems (e.g., from languages corresponding to infinite transition systems).
- Dynamic step firing policies where the policy applied depends on the current net marking (we do know how to deal with these in the case of PT-net systems and reward functions with marking dependent positive linear coefficients). Note that dynamic policies could become an important analytical tool for applications in the areas such as biochemistry and economics (for example, priority relations can be functions of the current state).
- Lookahead step firing policies where one decides at the current marking on the enabling of a step only after checking some possible future behaviours. We feel that such policies need to be treated with care as an arbitrary lookahead can be rather unrealistic (for example, deadlock prevention and enforced liveness could be coded as a policy). However, bounded *depth* lookahead policies could be interesting.

Finally, as we discovered while working on this paper, the notion of regions induced by morphisms proves to be surprisingly robust since it still provides a general characterisation of realisable transition systems in the case of Petri nets with step firing policies.

Acknowledgements. We are grateful to the reviewer who pointed out inconsistencies in the first version of this paper.

References

1. Badouel, E., Bernardinello, L., Darondeau, Ph.: Polynomial Algorithms for the Synthesis of Bounded Nets. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995)
2. Badouel, E., Darondeau, P.: On the Synthesis of General Petri Nets. Report INRIA-RR 3025 (1996)
3. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
4. Bernardinello, L., De Michelis, G., Petruni, K., Vigna, S.: On the Synchronic Structure of Transition Systems. In: Desel, J. (ed.) Structures in Concurrency Theory, pp. 11–31. Springer, Heidelberg (1996)
5. Busi, N., Pinna, G.M.: Synthesis of Nets with Inhibitor Arcs. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 151–165. Springer, Heidelberg (1997)
6. Chernikova, N.: Algorithm for Finding a General Formula for the Non-negative Solutions of a System of Linear Inequalities. USSR Computational Mathematics and Mathematical Physics 5, 228–233 (1965)

7. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. on Information and Systems* E80-D, 315–325 (1997)
8. Darondeau, Ph., Koutny, M., Pietkiewicz-Koutny, M., Yakovlev, A.: Synthesis of Nets with Step Firing Policies. Report CS-TR-1080, Newcastle University (2008)
9. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. *Acta Informatica* 33, 297–315 (1996)
10. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures; Part I: Basic Notions and the Representation Problem. *Acta Informatica* 27, 315–342 (1990)
11. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures; Part II: State Spaces of Concurrent Systems. *Acta Informatica* 27, 343–368 (1990)
12. INA, <http://www2.informatik.hu-berlin.de/~starke/ina.html>
13. Kleijn, J., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *WMC 2005*. LNCS, vol. 3850, pp. 292–309. Springer, Heidelberg (2006)
14. Koutny, M., Pietkiewicz-Koutny, M.: Transition Systems of Elementary Net Systems with Localities. In: Baier, C., Hermanns, H. (eds.) *CONCUR 2006*. LNCS, vol. 4137, pp. 173–187. Springer, Heidelberg (2006)
15. Koutny, M., Pietkiewicz-Koutny, M.: Synthesis of Elementary Net Systems with Context Arcs and Localities. In: Kleijn, J., Yakovlev, A. (eds.) *ICATPN 2007*. LNCS, vol. 4546, pp. 281–300. Springer, Heidelberg (2007)
16. Mazurkiewicz, A.: Petri Nets Without Tokens. In: Kleijn, J., Yakovlev, A. (eds.) *ICATPN 2007*. LNCS, vol. 4546, pp. 20–23. Springer, Heidelberg (2007)
17. Mukund, M.: Petri Nets and Step Transition Systems. *International Journal of Foundations of Computer Science* 3, 443–478 (1992)
18. Pietkiewicz-Koutny, M.: The Synthesis Problem for Elementary Nets with Inhibitor Arcs. *Fundamenta Informaticae* 40, 251–283 (1999)
19. Schrijver, A.: *Theory of Linear and Integer Programming*. John Wiley, Chichester (1986)
20. Starke, P.: Some Properties of Timed Nets under the Earliest Firing Rule. In: Rozenberg, G. (ed.) *APN 1989*. LNCS, vol. 424, pp. 418–432. Springer, Heidelberg (1990)

Modelling and Analysis of the INVITE Transaction of the Session Initiation Protocol Using Coloured Petri Nets

Lay G. Ding and Lin Liu

School of Computer and Information Science
University of South Australia

Mawson Lakes, SA 5095, Australia

dinlg001@students.unisa.edu.au, lin.liu@unisa.edu.au

Abstract. The Session Initiation Protocol (SIP) is a control protocol developed by the Internet Engineering Task Force for initiating, modifying and terminating multimedia sessions over the Internet. SIP uses an INVITE transaction to initiate a session. In this paper, we create a Coloured Petri Net (CPN) model for the INVITE transaction. Then we verify the general properties of the INVITE transaction by analysing the state space of the CPN model. The analysis results show that in most cases the INVITE transaction behaves as expected. However, in some circumstances, the transaction may terminate in an undesirable state while one communication party is still waiting for a response from its peer. Hence, we propose a set of changes to the INVITE transaction to correct the above problem. The result has shown that this revised INVITE transaction satisfies the properties that we have specified, and the undesirable terminal state has been eliminated.

Keywords: Session Initiation Protocol, Coloured Petri Nets, protocol verification.

1 Introduction

The popularisation of the Internet has been changing the way of communication in our daily life. A common example is the use of Voice over IP (VoIP). Before a conversation can take place between participants, protocols must be employed to establish a session, then to maintain and terminate the session. The Session Initiation Protocol (SIP) [1] is one of the protocols being used for such purposes.

SIP is developed by the Internet Engineering Task Force (IETF) and published as Request for Comments (RFC) 3261 in 2002 [1]. Besides its increasing popularity and importance in VoIP applications, SIP has been recognised by the 3rd Generation Partnership Project as a signalling protocol and permanent element of the IP Multimedia Subsystem architecture [2]. Thus, it is important to assure that the contents of RFC 3261 are correct, unambiguous, and easy to understand. Modelling and analysing the specification using formal methods can help in achieving this goal. Moreover, from the perspective of protocol

engineering, verification is also an important step of the life-cycle of protocol development [3,4], as a well-defined and verified specification will reduce the cost for implementation and maintenance.

SIP is a transaction-oriented protocol that carries out tasks through different transactions. Two main SIP transactions are defined [1], the INVITE transaction and the non-INVITE transaction. In this paper, we aim to verify the INVITE transaction, and consider only the operations over a reliable transport medium. Additionally, this paper will focus on functional correctness of the protocol, thus analysis of performance properties, such as session delay, is beyond its scope. Coloured Petri Nets (CPNs), their state space analysis method and supporting tools have been applied widely in verifying communication protocols, software, military systems, business processes, and some other systems [5,6,7]. However, to our best knowledge, very little work has been published on analysing SIP using CPNs, and only the study of [9,10] have been found. Most of the publications related to SIP are in the areas of interworking of SIP and H.323 [8], and SIP services [9,10]. In [9], the authors have modelled a SIP-based discovery protocol for the Multi-Channel Service Oriented Architecture, which uses the non-INVITE transaction of SIP as one of its basic components for web services in a mobile environment. However, no analysis results have been reported on this SIP-based discovery protocol. In [10], the authors have modelled SIP with the purpose of analysing SIP security mechanism, and have verified the CPN model in a typical attack scenario using state space analysis.

The rest of the paper is organised as follows. Section 2 is an overview of SIP layers and the INVITE transaction. Modelling and analysis of the SIP INVITE transaction are described in Section 3. Section 4 proposes and verifies the revised SIP INVITE transaction. Finally, we conclude the work and suggest future research in Section 5.

2 The INVITE Transaction of SIP

2.1 The Layered Structure of SIP

SIP is a layered protocol, comprising the syntax and encoding layer, transport layer, transaction layer, and transaction user (TU) layer, i.e. the four layers within the top box of Fig. 1.

The syntax and encoding layer specifies the format and structure of a SIP message, which can be either a request from a client to a server, or a response from a server to a client. For each request, a method (such as INVITE or ACK) must be carried to invoke a particular operation on a server. For each response, a status code is declared to indicate the acceptance, rejection or redirection of a SIP request, as shown in Table 1.

The SIP transport layer defines the behaviour of SIP entities in sending and receiving messages over the network. All SIP entities must contain this layer to send/receive messages to/from the underlying transport medium.

On top of SIP transport layer is the transaction layer, including the INVITE transaction and the non-INVITE transaction. An INVITE transaction is initi-

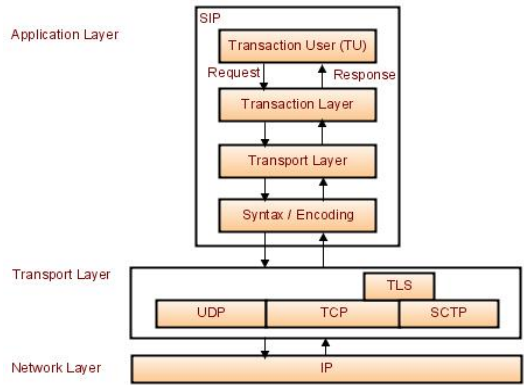


Fig. 1. Layered structure of SIP

Table 1. SIP response messages [1]

Response	Function
1xx (100-199)	Provisional - the request was received but not yet accepted
2xx	Success - the request was received successfully and accepted
3xx	Redirection - a further action is required to complete the request
4xx	Client Error - bad syntax found in the request
5xx	Server Error - the server failed to answer the request
6xx	Global Failure - no server can answer the request

ated when an INVITE request is sent; and a non-INVITE transaction is initiated when a request other than INVITE or ACK is sent. Each of the INVITE and non-INVITE transactions consists of a client transaction sending requests and a server transaction responding to requests.

Residing in the top layer of SIP are the TUs, which can be any SIP entity except a stateless proxy [1].

Among the four SIP layers, the transaction layer is the most important layer since it is responsible for request-response matching, retransmission handling with unreliable transport medium, and timeout handling when setting up or tearing down a session.

2.2 The INVITE Transaction

Operations of the client and the server transactions are defined in RFC 3261 by state machines and narrative descriptions. In this section we describe these in detail, where all states, timers, transport errors and responses are shown in *italics*, and all requests are capitalised.

INVITE Client Transaction. Referring to Fig. 2(a), an INVITE client transaction is initiated by the TU with an INVITE request. Meanwhile the client

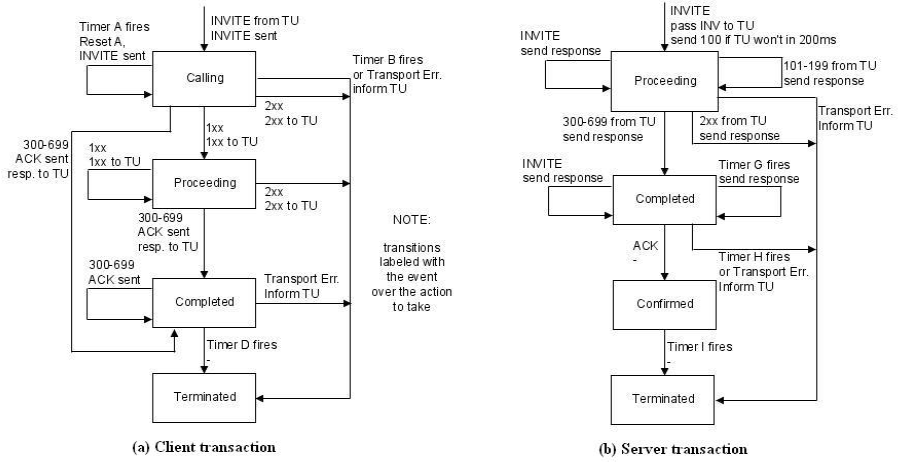


Fig. 2. INVITE client transaction (a) and INVITE server transaction (b)[1]

transaction enters its initial state, *Calling*¹. The INVITE request must be passed by the client transaction to SIP transport layer to be sent to the server side.

Once the *Calling* state is entered, *Timer B* is started for all transports (reliable or unreliable). For an unreliable transport medium, *Timer A* is also started, to control retransmissions of INVITE requests. While *Calling*, if the client transaction receives a *1xx* response (Table 1), it enters the *Proceeding* state. If a *Transport Err* (Error) occurs or *Timer B* expires, the client transaction moves to the *Terminated* state and informs its TU immediately. A *Transport Err* is indicated by SIP transport layer when a request cannot be sent to the underlying transport medium, which is generally caused by fatal ICMP errors in UDP or connection failures in TCP.

When in its *Proceeding* state, the client transaction may receive any number of provisional responses (*1xx*) before receiving a final response (*2xx* or *300-699*). While *Calling* or *Proceeding*, the reception of a final response by the client transaction will change its state to *Completed* or *Terminated*, depending on the type of the final response. If a *2xx* is received (indicating that the INVITE request is accepted by the server), the client transaction must enter its *Terminated* state, without sending any ACKs. If a *300-699* response is received (the call establishment was not successful), an ACK is generated by the client transaction and passed to SIP transport layer to be sent to the server side, and the client transaction moves to the *Completed* state. The reason for sending an ACK is to cease the retransmission of *300-699* responses by the server. All responses received by the client transaction when it is *Calling* or *Proceeding* must be passed to the TU except for retransmitted responses.

¹ Fig. 2(a) shows that an INVITE is sent before the *Calling* state is entered. However, based on the text description in RFC 3261, the *Calling* state is entered before the INVITE is sent to SIP transport layer. Additionally, the client transaction has to be in a state when sending a message. So we follow the text description in the RFC.

When the *Completed* state is entered, *Timer D* is started with a value of at least 32 seconds for an unreliable transport medium and zero seconds for a reliable medium. This timer is used to allow client transaction to absorb retransmitted *300-699* responses, and to re-pass ACKs to SIP transport layer. When the client transaction is in its *Completed* state, if a *Transport Err* occurs, it changes to the *Terminated* state and informs the TU of the failure of sending an ACK. If *Timer D* fires, the client transaction must also move to the *Terminated* state. Once the *Terminated* state is entered, the client transaction must be destroyed by its TU immediately.

INVITE Server Transaction. The server transaction is established by its TU at the server side when the TU receives a new INVITE request from the client side (Fig. 2 (b)). Once the server transaction is constructed, it enters the *Proceeding* state and sends a *100 Trying* response if the TU does not generate a response within 200 milliseconds², to cease retransmissions of INVITE requests by the client transaction.

While *Proceeding*, the server transaction can pass any provisional responses from its TU to SIP transport layer without changing state. If the response from its TU is a *2xx* response, or if it is informed of a transport error by SIP transport layer, the server transaction moves to its *Terminated* state. Otherwise, it will enter the *Completed* state, waiting for an acknowledgement from the client for the *300-699* response that was sent while in the *Proceeding* state. A transport error in the server transaction is indicated by SIP transport layer when a response cannot be sent to the underlying transport medium.

If a retransmitted INVITE is received in the *Proceeding* state, the most recent provisional response from TU must be passed to SIP transport layer for retransmission. If a retransmitted INVITE is received in the *Completed* state, the server transaction should pass a *300-699* response to SIP transport layer.

Once the *Completed* state is entered, *Timer H* is started. It sets the maximum time during which the server transaction can retransmit *300-699* responses. If the transport is unreliable, *Timer G* should also be started to control the time for each retransmission. If *Timer H* fires or a *Transport Err* occurs before an ACK is received by the server transaction, the transaction moves to its *Terminated* state. If an ACK is received before *Timer H* fires, the server transaction moves to its *Confirmed* state and *Timer I* is started with a delay of 5 seconds for an unreliable transport, and zero seconds for a reliable transport. *Timer I* is used to absorb additional ACKs triggered by the retransmission of *300-699* responses. When *Timer I* fires, *Terminated* state is entered. The server transaction is destroyed once it enters the *Terminated* state.

² Fig. 2(b) shows that if TU does not generate a response within 200ms, the server transaction must send a *100 Trying* response before the *Proceeding* state is entered. In fact, the INVITE server transaction cannot send any message before it is created (i.e. before entering the *Proceeding* state). This is inconsistent with the text description in the RFC 3261. So we follow the text description in the RFC.

3 Modelling and Analysis of the SIP INVITE Transaction

3.1 Modelling Assumptions

According to [1], SIP INVITE transaction can operate over a reliable (e.g. TCP) or an unreliable (e.g. UDP) transport medium. In this paper we assume a reliable transport medium is used, because firstly TCP is made mandatory in [1] for larger messages; secondly we use an incremental approach, checking whether the protocol works correctly over a perfect medium before checking its operations over an imperfect one. Furthermore, a lossy medium may mask some problems that will only be detected with a perfect medium.

Referring to Fig. 2(b), once the INVITE server transaction is created by its TU, if the transaction knows that the TU will not generate a response within 200 ms, it must generate and send a 100 Trying response. We assume that the INVITE server transaction does not know that the TU will generate a response within 200 ms after the server transaction is created, i.e. the server transaction must generate and send a 100 Trying response after it is created.

We also assume that a request message carries only a method (such as INVITE or ACK) and a response message carries only a status code (such as 200 OK), without including any header fields (such as Call-ID) or message bodies, as they are not related to the functional properties to be investigated.

3.2 INVITE Transaction State Machines with Reliable Medium

In Section 2 we have found some inconsistencies between the INVITE transaction state machines (Fig. 2) and the narrative descriptions given in Sections 17.1.1 and 17.2.1 of [1]. In this section, we present a revised version of the two state machines obtained by considering the modelling assumptions stated in the previous section and by eliminating the inconsistencies found in Section 2. We call these state machines “the INVITE transaction state machines with reliable transport medium” (Fig. 3), and the state machines provided in [1] (Fig. 2) “the original state machines”.

Referring to Fig. 2, a number of timers are defined in the original state machines to deal with message loss or processing/transmission delays. When the transport medium is reliable, the requests (INVITE and ACK) and the final responses (non-1xx) in the original INVITE server transaction are sent only once [1]. As a result, Timer A of the INVITE client transaction and Timer G of the INVITE server transaction are not applied (see Fig 3). Additionally, as Timer I of the original INVITE server transaction is set to fire in zero seconds for a reliable transport medium, the Confirmed state and Timer I are not considered. Hence, after an ACK for 300-699 response is received, the INVITE server transaction is terminated immediately (Fig. 3(b)).

Because we have assumed that the server transaction must generate and send a 100 Trying response after it is created, we remove the if clause “if TU won’t in 200ms” from the top of the original INVITE server transaction state machine. Additionally as noted in Section 2 (footnote 2), the INVITE server transaction

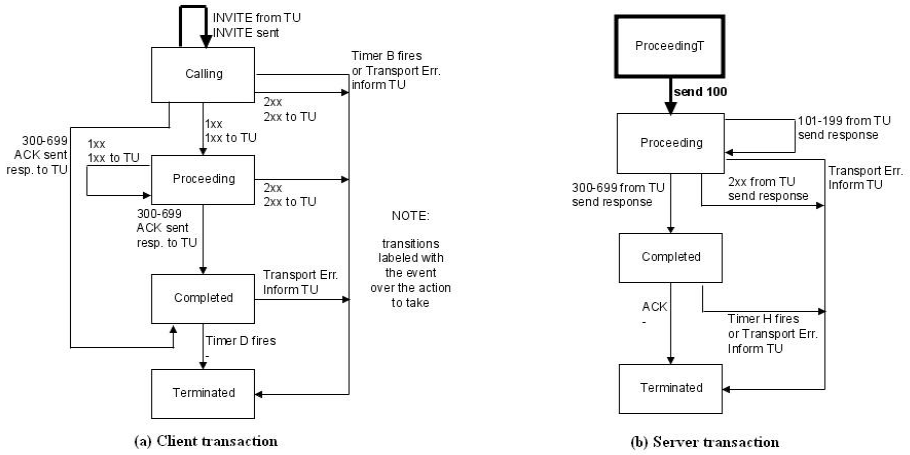


Fig. 3. INVITE client transaction (a) and server transaction (b) with reliable medium

can not receive or send any messages before it is created (i.e. before the Proceeding state is entered). So we firstly remove the action “pass INV to TU” from the top of the original state machine. Then to specify the state of the server transaction when it has just been created by its TU, we add a new state, *ProceedingT*. In the ProceedingT state, the only action to be carried out by the server transaction is to generate and send a 100 Trying response (see Fig. 3(b)).

A further modification is made to the original state machine for the INVITE client transaction based on the inconsistency mentioned in Section 2 (footnote 1). According to the narrative description provided in Section 17.1.1 of [1], the INVITE client transaction must firstly enter its Calling state to pass an INVITE received from its TU to SIP transport layer. However, the original state machine (Fig. 2(a)) shows that sending an INVITE by the client transaction can occur before it enters the Calling state (i.e. before the transaction is created), which is impossible. Therefore, we modified the input arc at the top of the original client state machine (refer to Fig. 2(a)) so that an INVITE request can be received from its TU and passed to the SIP transport layer by the INVITE client transaction only when the transaction is in its Calling state (see Fig. 3). Note that the event and action that label this input arc (to the Calling state) can not occur more than once due to the reason that the TU only passes one INVITE request to an INVITE client transaction [1].

3.3 CPN Model of the INVITE Transaction

The CPN model for the INVITE transaction is shown in Fig. 4 (declarations) and Fig. 5 (the CPN). This model is based on the state machines shown in Fig. 3. In the following, names of places, transitions, and variables of the CPN model are written in **typewriter** style. To distinguish a server transaction’s state from a client transaction’s state with the same name, a capitalised S is


```

▼ Declarations
▼ val n = 3;
▼ colset INT = int with 0..1;
▼ colset STATEC = with calling | proceeding | completed | terminated;
▼ colset REQUEST = with INVITE | ACK;
▼ colset REQUESTQ = list REQUEST;
▼ colset STATES = with Idle | proceedingT | proceedingS | completedS | terminatedS;
▼ colset RESPONSE = with r100 | r101 | r2xx | r3xx;
▼ colset Response = subset RESPONSE with [r101, r2xx, r3xx];
▼ colset RESPONSEQ = list RESPONSE;
▼ var a : INT;
▼ var sc : STATEC;
▼ var req : REQUEST;
▼ var requestQ : REQUESTQ;
▼ var ss : STATES;
▼ var responseQ : RESPONSEQ;
▼ var re : Response;
▼ var res : RESPONSE;

```

Fig. 4. Declarations of the CPN model of the INVITE transaction

appended to the name of the state of the server transaction (except for the `proceedingT` state). For example, `proceedingS` represents the Proceeding state of the server transaction while `proceeding` represents the Proceeding state of the client transaction. SIP response messages (Table 1) are named as follows: `r100` represents a 100 Trying response; `r101` is for a provisional response with a status code between 101 and 199; `r2xx` for a response with a status code between 200 and 299; and `r3xx` for a response with a status code between 300 and 699.

Declarations. Referring to Fig. 4, a constant, `n`, is defined to represent the maximum length of the queue in place `Responses` (Fig. 5). Four colour sets, `INT`, `STATEC`, `REQUEST`, and `REQUESTQ`, are declared for modelling the client transaction. `INT` is the set of integers between 0 and 1, typing place `INVITE Sent` where the number of INVITE requests that have been sent is counted. `STATEC` (typing place `Client`) models all the possible states of the INVITE client transaction. `REQUEST` models the two SIP methods specified for the INVITE transaction, INVITE and ACK. `REQUESTQ` is a list of `REQUEST` (typing place `Requests`). To model the server transaction, colour sets `STATES`, `RESPONSE`, `Response`, and `RESPONSEQ` are declared. `STATES` is used to type place `Server`. It defines all the possible states of the server transaction, and a temporary state, `Idle` (modelling the existence of the INVITE server transaction). `RESPONSE` models the four different categories of responses from the server side, and `Response` is a subset of `RESPONSE`, used in the inscriptions of the arcs associated with transition `Send Response` (see Fig. 5). This subset is used for modelling that any response except `r100` can be sent when the server is in its `proceedingS` state, which can be implemented using the variable `re` that runs over the subset. `RESPONSEQ` is a list of responses sent by the server transaction, and it is used to type place `Responses`, to model a First-In-First-Out queue. Variable `a` is of type `INT`, and `sc` and `ss` can take values from the colour sets `STATEC` and `STATES` respectively. Variables `req` and `res` are of types `REQUEST` and `RESPONSE` respectively. For dealing with the lists that store requests and responses, we declare variables `requestQ` of type `REQUESTQ` and `responseQ` of type `RESPONSEQ`.

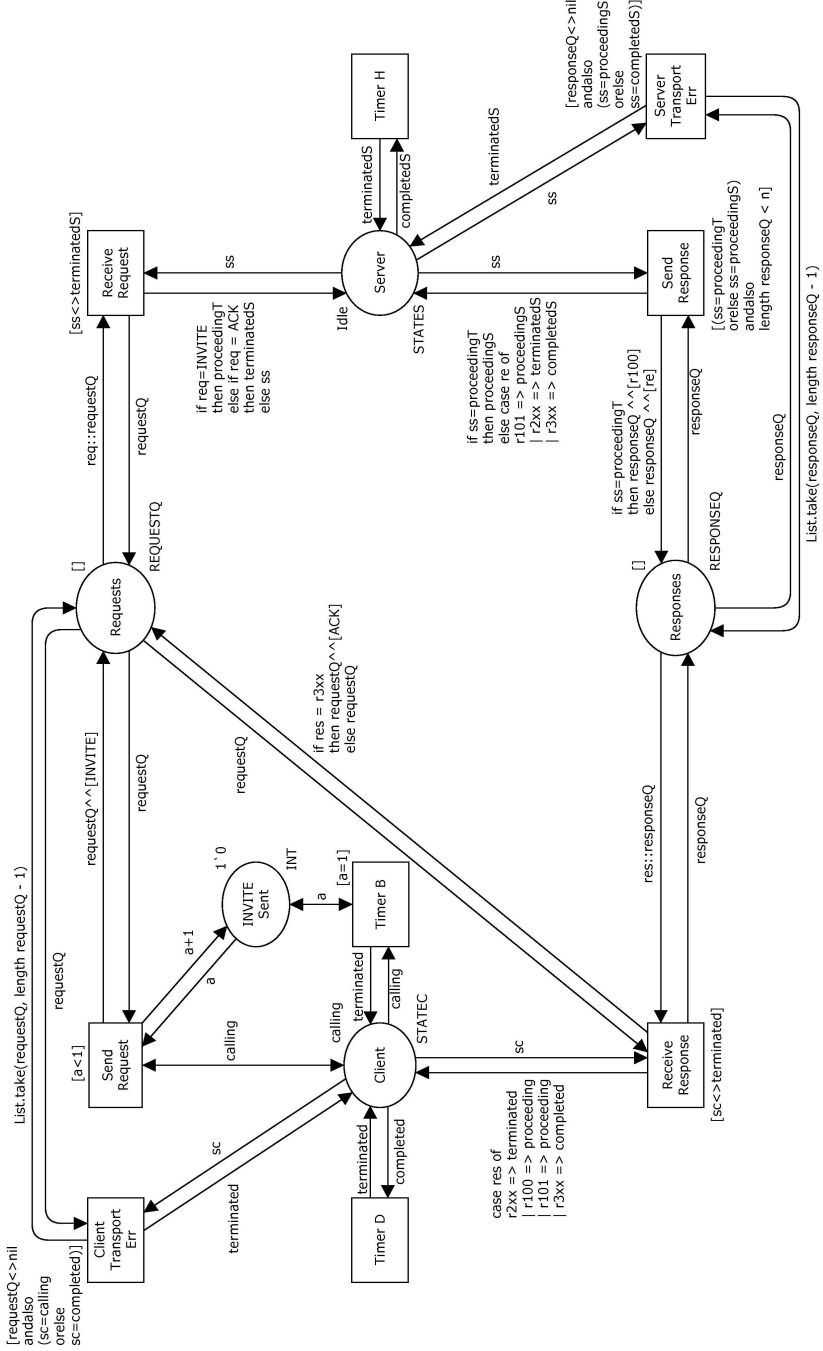


Fig. 5. CPN model of the INVITE transaction

INVITE Client Transaction. The left part of the CPN model (Fig. 5), including places **Client** and **INVITE Sent**, and the transitions connected to them, models the client transaction state machine with reliable medium (Fig. 3(a)).

States of the client transaction are represented by place **Client** (typed with colour set **STATEC**). The initial state of **Client** is **calling**, indicating that the INVITE client transaction has been initiated by its TU, and an INVITE request has been passed from its TU to the client transaction for transmission.

Five transitions are associated with **Client**, to model operations of the INVITE client transaction. **Send Request** models how the transaction passes an INVITE request received from its TU to SIP transport layer. It is enabled only if there is a **calling** in **Client** and no INVITE request has been passed to SIP transport layer (i.e. **INVITE Sent** contains an integer with value 0). **Receive Response** models how the client transaction receives responses and sends ACKs. It is enabled when a response is received (i.e. removed from the head of the queue in place **Responses**) and the **Client** is not **terminated**. If the client transaction receives a 300-699 response (**r3xx**), an ACK is passed to SIP transport layer, and **Client** moves to its **completed** state. If the received response is **r100**, **r101** or **r2xx**, no ACK is sent; when the response is **r100** or **r101**, **Client** moves to **proceeding**; and when the response is **r2xx**, the **Client** moves to **terminated**.

Timer B is modelled by transition **Timer B**. Since our focus is on the functional correctness of SIP rather than its performance properties such as session delay, values of timers are not modelled. To model that Timer B can not be started before an INVITE request is sent, place **INVITE Sent** is set to an input place of **Timer B**. **Timer B** is enabled only when an integer with value 1 is in **INVITE Sent** (see the guard **[a=1]**) and the **Client** is **calling**. The initial marking of **INVITE Sent** is 0 (no INVITE request has been sent to SIP transport layer), when **Send Request** is fired (i.e. an INVITE request has been sent), the integer value is incremented by 1.

Timer D (Fig. 3(a)) sets the maximum time for which the client transaction can stay in its **Completed** state to wait for retransmitted 300-699 responses from the server transaction. Since there are no retransmissions when the transport medium is reliable, according to [1], Timer D is set to fire in zero seconds in this case. Once it fires, the client transaction enters its **Terminated** state. This seems to indicate that the client transaction would enter the **Terminated** state immediately after it is in the **Completed** state, and nothing else can happen in between the two states. Thus we might be able to fold the two states into one and not to consider Timer D when the transport is reliable. However, from Fig. 3, a **Transport Err** can occur when the client transaction is in its **Completed** state, and the **Completed** state is entered after the client transaction has passed an ACK to SIP transport layer. The transport layer may report an error immediately when it receives the ACK, thus a **Transport Err** occurs at the transaction layer. According to Fig. 3, the transaction layer needs to inform its TU of this error when it is in the **Completed** state. From this perspective, we can not fold the **Completed** and **Terminated** states. Therefore, we create a transition, **Timer**

D. It is enabled once there is a **completed** in **Client**, and its occurrence brings **Client** to **terminated**.

Transition **Client Transport Err** (modelling a Transport Err at the transaction layer) is enabled when the **Client** is **completed**, and its occurrence also brings **Client** to **terminated**. When an error is reported by SIP transport layer, the ACK that has been passed to it from the transaction layer will not be sent to the server side, so when **Client Transport Err** occurs, the ACK that has just been put in the queue in **Requests** is destroyed (see the inscription of the arc from **Client Transport Err** to **Requests**). From Fig. 3(a), a Transport Err can occur when the client transaction is **Calling**, so **Client Transport Err** can be enabled as well when a **calling** is in **Client** (see the guard of **Client Transport Err**).

INVITE Server Transaction. Referring to the right part of Fig. 5, place **Server** and the four transitions connected to it model the INVITE server transaction specified in Fig. 3(b). Place **Server** models the states of the transaction, **proceedingT**, **proceedingS**, **completedS**, **terminatedS**. **Idle** (see Fig. 4) is not a state of the server transaction, it is the initial marking of **Server**, indicating that it is ready for the TU to create a server transaction once the TU receives an INVITE request. Transition **Receive Request** models the reception of a request (INVITE or ACK). While there is an **Idle** in **Server**, if the request received is an INVITE, a **proceedingT** is created in **Server**, modelling that the server transaction is created (by the TU) and it is now in its **proceedingT** state (in this case and only in this case, transition **Receive Request** models the operation of the TU instead of the server transaction of receiving an INVITE request from the client side). Once the **Server** enters its **proceedingT** state, **Send Response** is enabled, thus **r100** can be placed into the queue in **Responses**, changing the state of the **Server** to **proceedingS**. In the **proceedingS** state, **Send Response** is again enabled. When it occurs, either a provisional response **r101** or a final response (**r2xx** or **r3xx**) is appended to the queue in **Responses**, and a **proceedingS** (if a **r101** is put in the queue), **completedS** (for **r3xx**) or **terminatedS** (for **r2xx**) is put in **Server** (refer to the **else** clauses of the inscriptions of the outgoing arcs of **Send Response**). While the **Server** is **completedS**, if an ACK is received, the occurrence of **Receive Request** will generate a **terminatedS** in the **Server**. The guard of **Receive Request** models that the server transaction can not receive any requests after it enters the **Terminated** state because a server transaction is destroyed immediately after the **Terminated** state [1].

Only one timer, **Timer H**, is used by the INVITE server transaction (Fig. 3(b)) when the transport medium is reliable, to control the transmission and retransmission of 300-699 responses by the INVITE server transaction. It is modelled by transition **Timer H**, which is enabled when the **Server** is **completedS**. When it fires, **Server** moves to its **terminatedS** state, indicating that an ACK corresponding to a 300-699 response is never received by the server transaction before **Timer H** has expired. **Server Transport Err** models a transport error occurring at the server side. It is enabled after a response has been sent to SIP transport layer (i.e. when the server transaction is **proceedingS** or **completedS**). When it

fires, the response that has just been put into the queue in **Responses** is removed (see the inscription of the arcs from **Server Transport Err** to **Responses**, the **List.take** function returns the remained responses of list **responseQ**) and a **terminatedS** is put into **Server**.

The Underlying Transport Medium. The middle part of the CPN model (i.e. places **Requests** and **Responses** of Fig. 5) models SIP transport layer and the underlying transport medium (see Fig. 1). **Requests** models the transmission of requests from the client side to the server side; whereas **Responses** models the transmission of responses in the reverse direction. The maximum number of provisional responses (i.e. **r101**) that can be sent from the server transaction is not given in [1], so there may be an arbitrarily large number of **r101** to be put into the queue in **Responses**. We use a positive integer parameter, **n** (Fig. 4) to represent the maximum length of the queue in **Responses** and a guard for transition **Send Response** (**[length responseQ < n]**) to limit the maximum length of the queue in **Responses**.

3.4 State Space Analysis of the INVITE Transaction CPN Model

In this section, we firstly define the desired properties for the INVITE transaction, then we analyse the state space of the CPN model described in the previous section. In order to avoid state explosion problem with state space analysis, we use 3 as the maximum length of the queue in place **Responses** (i.e. **n=3**).

The functional properties that we are interested in include *absence of deadlocks* and *absence of livelocks*. According to [3] a protocol can fail if any of the two properties are not satisfied. We also expect that the INVITE transaction has *no dead code* (and action that is specified but never executed). A deadlock is an undesired dead marking in the state space of a CPN model, and a marking is dead if no transitions are enabled in it [11]. For the INVITE transaction, there is only one desired dead marking, representing the desirable terminal state of the INVITE transaction. In this state both the client and the server transactions are in their Terminated state (see Fig. 3), and no messages remain in the communication channel. Any other dead marking is thus undesirable, i.e. a deadlock. A livelock is a cycle of the state space that once entered, can never be left, and within which no progress is made with respect to the purpose of the system.

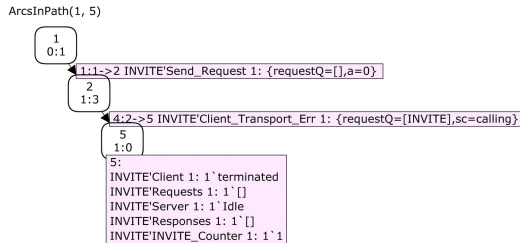
To analyse the desired properties of the INVITE transaction, we firstly check the state space report generated by CPN Tools [12]. The report shows that a full state space with 52 nodes and 103 arcs is generated. There are 3 nodes fewer in the Strongly Connected Components (SCC) graph than in the state space, which means that the state space contains cycles (which needs to be further investigated to see if they are livelocks). We also found ten dead markings in the state space, which are nodes 5, 19, 25, 28, 29, 30, 36, 45, 46 and 50 (Table 2). Additionally, there are no dead transitions in the state space, so the INVITE transaction has no dead code.

From Table 2, we can see that node 25 is a desirable dead marking. The requests and responses have been sent and received. Both the client and server

Table 2. List of dead markings

Node	Client	Requests	Server	Responses	INVITE Sent
5	terminated	[]	Idle	[]	1'1
19	terminated	[]	terminatedS	[r100, r2xx]	1'1
25	terminated	[]	terminatedS	[]	1'1
28	proceeding	[]	terminatedS	[]	1'1
29	terminated	[]	terminatedS	[r100, r3xx]	1'1
30	terminated	[]	terminatedS	[r100]	1'1
36	terminated	[]	terminatedS	[r100, r101, r2xx]	1'1
45	terminated	[]	terminatedS	[r100, r101, r3xx]	1'1
46	terminated	[]	terminatedS	[r100, r101]	1'1
50	terminated	[ACK]	terminatedS	[]	1'1

transactions are in their Terminated states. Moreover, no messages remain in the channel, i.e. places **Requests** and **Responses** each has an empty list. At node 5 the server's state is **Idle**, which is different from other dead markings (i.e. **terminatedS**). To find out whether this node is an undesirable dead marking, we use CPN Tools' query **ArcsInPath (1, 5)** to generate the path to this dead marking (Fig. 6). At the initial state, node 1, the client transaction sends an INVITE request to SIP transport layer. However, a transport error occurs at node 2 and no request is sent to the server side. Hence, the client transaction is terminated and no corresponding INVITE server transaction is constructed at the server side (node 5). This behaviour is expected [1]. Furthermore, for each of the nodes 19, 29, 30, 36, 45, 46 and 50 in Table 2, there are still messages remaining in the communication channel after both the client and server transactions have been terminated. These dead markings are caused by either transport error or timeout. Once a transport error or timeout occurs, the transaction will be destroyed. Therefore a message left in the channel will find no matching transaction. However, these dead markings are acceptable, because if the message is a response, according to [1], the response must be silently discarded by SIP transport layer of the client. If the message is a request (i.e. node 50), according to [1], the request (i.e. ACK) will be passed to the TU of the destroyed server transaction to be handled.

**Fig. 6.** The path from Node 1 to Node 5

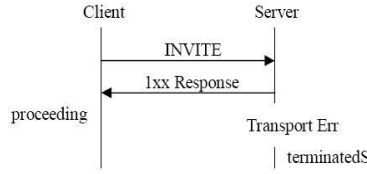


Fig. 7. Scenario of undesired behaviour

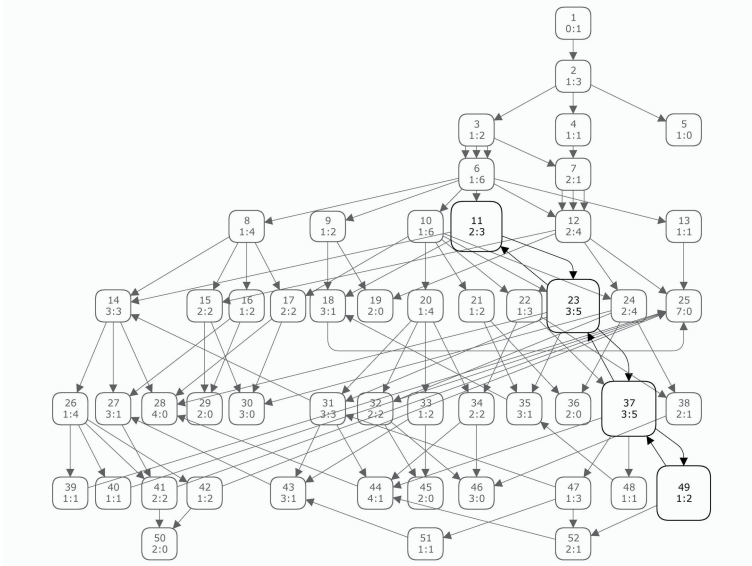


Fig. 8. State Space of the INVITE transaction CPN model

However, node 28 is an undesired dead marking (deadlock). Referring to Table 2, with node 28, the state of the client transaction is **proceeding**, while the server transaction is **terminatedS** due to a transport error (Fig. 7). This behaviour is not expected since in this case the server transaction has been destroyed (i.e. no responses can be received by the client transaction). No timer is given in [1] to specify the maximum time the client transaction can wait for a final response when it is in the **proceeding** state. Thus, when a transport error occurs at the server side, the client transaction will have no way to come out from the **proceeding** state.

To check if there are livelocks, we draw the state space of the INVITE transaction (Fig. 8). As indicated by the difference between the size of the state space and the size of the SCC graph, there are cycles in the state space: the cycles between nodes 11, 23, 37 and 49. However, because from any of the four nodes,

the transaction can move to a state which is not within a cycle, these cycles are not livelocks. Therefore, the INVITE transaction is free of livelocks.

4 The Revised INVITE Transaction and Its Verification

4.1 Changes to the INVITE Transaction

In the previous section, we have found that the INVITE transaction can reach a deadlock where the server transaction has terminated but the client transaction is still in its Proceeding state. After discovering this using state space analysis of CPNs, we noticed that this deadlock had been discussed and reported by SIP implementers [13]. However, so far there is no solution published for fixing this bug. In this section, we propose a modification to the INVITE transaction to eliminate the deadlock.

In [1], it is recommended that, at the TU level, the TU at the client side should destroy the original INVITE client transaction, if the TU does not receive a final response within $64 \times T1$ seconds ($T1 = 500\text{ms}$) after an INVITE request is passed to the transaction layer. So for an application that implements this recommendation, the INVITE client transaction will be destroyed eventually if the transaction has reached the deadlock. However, with applications that do not implement this recommendation, a caller will not receive any indications from the system, thus may wait for a long time (listening to the ringing tone) before giving up (i.e. hanging up the phone). Therefore we believe that it is necessary to solve this problem at the transaction layer, i.e. to force the client transaction to reach its Terminated state, instead of staying in the Proceeding state, so that the TU can destroy the transaction and avoid an unnecessarily long wait.

From Fig. 3(a) we see that the client transaction can go from the Proceeding state to the Completed state then the Terminated state only if it receives a final response (300-699 or 2xx) from the server transaction. However, after having reached the deadlock, no responses can be received by the client transaction since the server transaction has been terminated. Therefore, in this case, to bring the client transaction to its Terminated state, we need to use an event that occurs at the client side to trigger this state change, i.e. a timer for the Proceeding state. Referring to Fig. 9, before the client transaction enters the Proceeding state, it now needs to reset Timer B (i.e. restarts it with value $64 \times T1$ ms). Then in the Proceeding state, once Timer B expires, the INVITE client transaction notifies the TU about the timeout and moves to the Terminated state.

4.2 The Revised INVITE Transaction CPN and Its Analysis

Fig. 10 shows the CPN model for the revised INVITE transaction. It is obtained from the CPN in Fig. 5 by modifying the arc inscription and the guard of transition **Timer B**. The arc inscription from place **Client** to **Timer B** has been changed from **calling** to a variable, **sc**, of colour set **STATEC**. This variable can be bound to any value of **STATEC**, but **Timer B** should not be enabled in states other than **calling** or **proceeding**. Therefore, the guard of **Timer B**

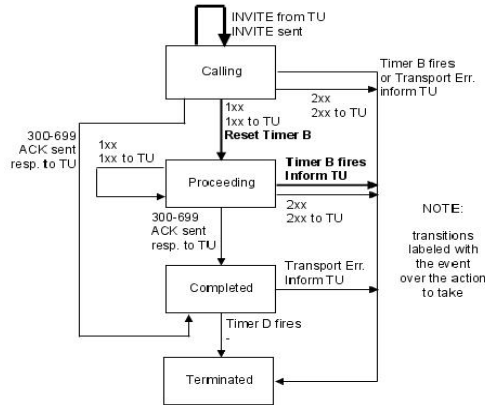


Fig. 9. Revised INVITE client transaction state machine

has also been changed. In Fig. 5, the guard **Timer B** (`[a=1]`), is used to set the condition that **Timer B** can not be enabled before an **INVITE** request has been sent in the **calling** state. Since **proceeding** is not associated with sending the **INVITE** request, the guard for the revised CPN model is modified to `[(sc=calling andalso a=1) orelse sc=proceeding]`.

The same as with analysing the original CPN model shown in Fig. 5, we use 3 as the maximum length of the queue in place **Responses** (i.e. **n=3**) to avoid state explosion. The state space of the CPN model is then generated. The state

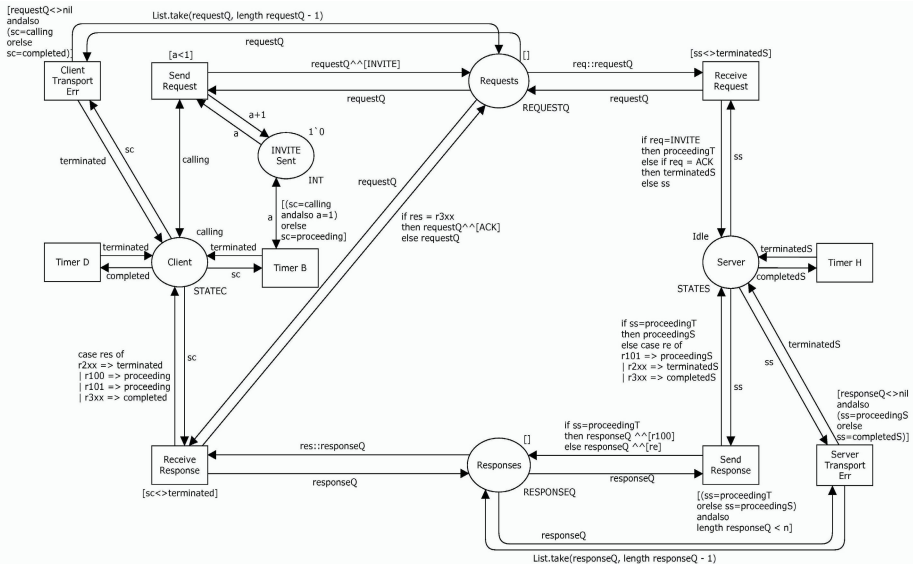


Fig. 10. CPN model of the revised INVITE transaction

space report shows that there are more nodes and arcs generated for both the state space and SCC graph. However, the differences between the state space and SCC graph, i.e. 3 nodes and 6 arcs, have remained the same. Additionally, the report shows that there are no dead transitions in the state space of the INVITE transaction, so it has no dead code.

All the dead markings are shown in Table 3. We see that the deadlock in the state space of the original model (node 28 of Table 2) has been removed. However, the state space of the revised CPN model has more dead markings (i.e. nodes 59, 60, 64, 66 and 67). This is because in the Proceeding state of the INVITE client transaction (refer to Fig. 10), when **Timer B** occurs, the **Client** moves to **terminated** before the responses (i.e. **r101**, **r2xx** or **r3xx**) have been received (queuing in **Responses**). Previously, we only had these responses queuing in the **Responses** when **Client** was marked by **proceeding**. Since these dead markings (nodes 59, 60, 64, 66 and 67) have similar behaviour to nodes 19, 26, 31, 32, 33, 39, 47, 51, 52, 53, 58 (see Table 3), i.e. messages are left in the channel, and client and server transactions have each entered its Terminated state, they are all acceptable. Furthermore, from Table 3, nodes 5 and 26 were discovered in the state space of the original model, and they have already been identified as desirable dead markings of the INVITE transaction.

To check if there are livelocks in the revised INVITE transaction, we draw the state space (Fig. 11). There are cycles between nodes 11, 23, 40 and 56. However, none of them are livelocks because from each of the four nodes the transaction can move to a state that is not within a cycle. Therefore, the revised INVITE transaction has no livelock.

Table 3. List of dead markings of the revised CPN model

Node	Client	Requests	Server	Responses	INVITE Sent
5	terminated	[]	Idle	[]	1'1
19	terminated	[]	terminatedS	[r100, r2xx]	1'1
26	terminated	[]	terminatedS	[]	1'1
31	terminated	[]	terminatedS	[r100, r3xx]	1'1
32	terminated	[]	terminatedS	[r100]	1'1
33	terminated	[]	terminatedS	[r2xx]	1'1
39	terminated	[]	terminatedS	[r100, r101, r2xx]	1'1
47	terminated	[]	terminatedS	[r3xx]	1'1
51	terminated	[]	terminatedS	[r100, r101, r3xx]	1'1
52	terminated	[]	terminatedS	[r100, r101]	1'1
53	terminated	[]	terminatedS	[r100, r2xx]	1'1
58	terminated	[ACK]	terminatedS	[]	1'1
59	terminated	[]	terminatedS	[r101, r3xx]	1'1
60	terminated	[]	terminatedS	[r101]	1'1
64	terminated	[]	terminatedS	[r101, r101, r2xx]	1'1
66	terminated	[]	terminatedS	[r101, r101, r3xx]	1'1
67	terminated	[]	terminatedS	[r101, r101]	1'1

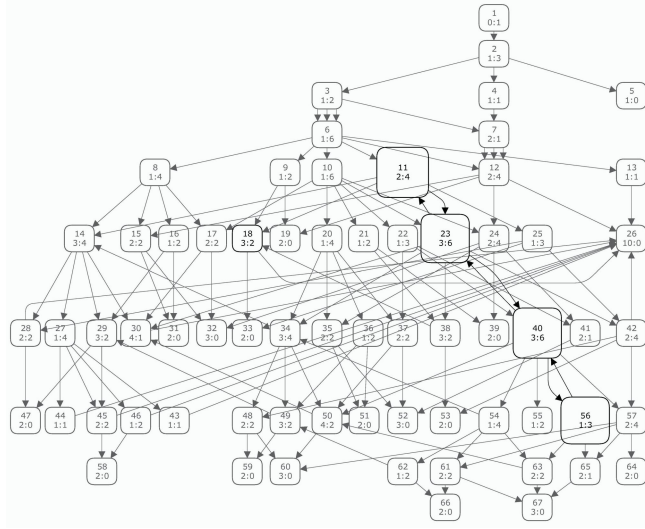


Fig. 11. Cycles in the state space of the revised INVITE transaction CPN model

5 Conclusions and Future Work

The INVITE transaction is one of the essential transactions of SIP. It has been used in conjunction with other protocols to establish sessions and provide communication services. Based on the state machines and narrative descriptions that are provided in [1], we have modelled and analysed the SIP INVITE transaction with reliable transport medium using CPNs. The contributions of the paper are summarised below.

- Refinement to the definition of the INVITE transaction. We have found some inconsistencies between the state machines and the narrative descriptions in [1]. Modifications have been proposed to the state machines to remove the inconsistencies. After omitting the states and actions which are defined for SIP over an unreliable transport medium only, we have obtained the state machines for the INVITE transaction over a reliable transport medium, and have created a CPN model for it, which provides a formal specification for the INVITE transaction.
- Verification of the INVITE transaction. By examining the state space of the CPN model, we have found that the INVITE transaction has no livelock or dead code. We have also found in the state space of the INVITE transaction a sequence of events that lead to the desirable terminal state, however, the INVITE transaction may terminate in an undesirable state, in which the INVITE client transaction is still in its Proceeding state.
- Revision to the definition of the INVITE transaction and its verification. To eliminate the undesirable behaviour, we have proposed a set of changes to the INVITE client transaction. Using state space analysis, we have found that the revised INVITE transaction has satisfied the desired properties.

In the future, we shall model and analyse the INVITE transaction with unreliable transport medium. We have noticed that, very recently, an Internet draft (work in progress) has been published by IETF, to propose updates to the INVITE transaction state machines [14]. The proposed updates have no impacts on the behaviour of the INVITE transaction when the transport medium is reliable, which means IETF may have not been aware of the incompleteness of [1] of the specification of the INVITE transaction. On the other hand, the proposed updates may have influence on the INVITE transaction when the transport medium is unreliable. Therefore, the other possible future work can include modelling and analysing the updated version of INVITE transaction proposed in the Internet Draft [14]. In this way, the correctness of the proposed updates given in the Internet Draft [14] can be checked and confirmed.

Acknowledgements. We would like to express our appreciation to the members of Computer Systems Engineering Centre, University of South Australia, for attending our presentation and providing useful comments on our research. Especially, we would like to thank Professor Jonathan Billington, Dr Guy Gallasch, and Dr Somsak Vanit-Anunchai for all their helpful suggestions and invaluable comments.

References

1. Rosenberg, J., et al.: RFC 3261: SIP: Session Initiation Protocol. Internet Engineering Task Force (2002), <http://www.faqs.org/rfcs/rfc3261.html>
2. Sparks, R.: SIP: basics and beyond. Queue 5(2), 22–33 (2007)
3. Holzmann, G.J.: Design and validation of computer protocols. Prentice Hall, Englewood Cliffs, New Jersey (1991)
4. Sidhu, D., Chung, A., Blumer, T.P.: Experience with formal methods in protocol development. In: ACM SIGCOMM Computer Communication Review, vol. 21(2), pp. 81–101. ACM, New York (1991)
5. Examples of Industrial Use of CP-nets, http://www.daimi.au.dk/CPnets/intro/example_indu.html
6. Billington, J., Gallasch, G.E., Han, B.: Lectures on Concurrency and Petri Nets: A Coloured Petri Net Approach to Protocol Verification. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 210–290. Springer, Heidelberg (2004)
7. Kristensen, L.M., Jørgensen, J.B., Jensen, K.: Application of Coloured Petri Nets in System Development. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 626–685. Springer, Heidelberg (2004)
8. Turner, K.J.: Modelling SIP Services Using CRESS. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 162–177. Springer, Heidelberg (2002)
9. Gehlot, V., Hayrapetyan, A.: A CPN Model of a SIP-Based Dynamic Discovery Protocol for Webservices in a Mobile Environment. In: the 7th Workshop and Tutorial on Practical Use of CPNs and the CPN Tools, University of Aarhus, Denmark (2006)
10. Wan, H., Su, G., Ma, H.: SIP for Mobile Networks and Security Model. In: Wireless Communications, Networking and Mobile Computing, pp. 1809–1812. IEEE, Los Alamitos (2007)

11. Jensen, K., Kristensen, L., Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. on Software Tools for Technology Transfer (STTT)* 9(3), 213–254 (2007)
12. Home Page of the CPN Tools,
<http://wiki.daimi.au.dk/cpntools/cpntools.wiki>
13. Rosenberg, J.: Bug 706 - Clarify lack of a timer for exiting proceeding state, Bugzilla (2003), http://bugs.sipit.net/show_bug.cgi?id=706
14. Sparks, R.: draft-sparks-sip-invfif-00: Correct transaction handling for 200 responses to Session Initiation Protocol INVITE requests. Internet Engineering Task Force (2007), <http://tools.ietf.org/id/draft-sparks-sip-invfif-00.txt>

Modelling and Initial Validation of the DYMO Routing Protocol for Mobile Ad-Hoc Networks*

Kristian L. Espensen, Mads K. Kjeldsen, and Lars M. Kristensen

Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark,
{espensen,kebløv,kris}@daimi.au.dk

Abstract. A mobile ad-hoc network (MANET) is an infrastructureless network established by a set of mobile devices using wireless communication. The Dynamic MANET On-demand (DYMO) protocol is a routing protocol for multi-hop communication in MANETs currently under development by the Internet Engineering Task Force (IETF). This paper presents a Coloured Petri Net (CPN) model of the mandatory parts of the DYMO protocol, and shows how scenario-based state space exploration has been used to validate key properties of the protocol. Our CPN modelling and verification work has spanned two revisions of the DYMO protocol specification and have had direct impact on the most recent version of the protocol specification.

Keywords: System design and verification using nets, experience with using nets, case studies, higher-level net models, application of nets to protocols and networks.

1 Introduction

Mobile ad-hoc networks (MANETs) [14] is a networking paradigm motivated by the increased presence and use of mobile wireless networking devices such as laptops, PDAs, and mobile phones. The basic idea of MANETs is that a set of mobile nodes is able to autonomously establish a wireless network without relying on a preexisting communication infrastructure (such as the Internet). A typical application of MANETs is emergency search-and-rescue operations in remote areas where no preexisting communication infrastructure is available. The nodes in a MANET are characterised by having limited memory, processing power, and battery capacity. Furthermore, the topology of a MANET changes frequently due to the mobility of the nodes and the varying quality of the wireless links. These characteristics imply that communication protocols for conventional infrastructured networks in most cases are not suited for MANETs. The central communication service of a MANET is multi-hop routing which enables the nodes to forward data packets, and several routing protocols [14,11] are currently under development.

* Supported by the Danish National Research Council for Technology and Production.

In this paper we consider the Dynamic MANET On-demand (DYMO) [2] routing protocol being developed by the IETF MANET working group [11]. The DYMO protocol specification [2] is currently an Internet-draft in its 11th revision and is expected to become a Request for Comments (RFC) document in the near future. The recent discussions on the mailing list [12] of the MANET working group have revealed several complicated issues in the DYMO protocol specification, in particular related to the processing of routing messages and the handling of sequence numbers. This, combined with the experiences of our research group in implementing the DYMO protocol [15,16] and conducting initial modelling [9], has motivated us to initiate a project [7] aiming at constructing a Coloured Petri Net (CPN) model [10] of the DYMO protocol specification using CPN Tools [5] with the goal of validating the correctness of the protocol using state space exploration. Next, via a set of refinement steps we will use the CPN model (preferably via automatic code generation) as a basis for implementing the routing daemon of the DYMO protocol. This paper presents our modelling and initial validation work in this project.

We present a CPN model of mandatory parts of the DYMO protocol. In addition to modelling DYMO, the modelling of the wireless mobile network is aimed at being generally applicable for modelling MANET protocols. Finally, we present state space exploration results concerning the establishment of routes and processing of routing messages. Our modelling and validation work has spanned revision 10 [3] and revision 11 [2] of the DYMO specification. Our work on revision 10 identified several ambiguities and possible improvements which were submitted [12], acknowledged by the DYMO working group, and taken into account in revision 11.

This paper is structured as follows. Section 2 gives a brief introduction to the basic operation of the DYMO protocol. Section 3 presents the CPN model and provides additional details about the DYMO protocol. In Sect. 4 we present initial investigations of the behaviour of the DYMO protocol using state space exploration. Finally, in Sect. 5 we summarise our conclusions and discuss related and future work. The reader is assumed to be familiar with the basic ideas of the CPN modelling language [10] and state space exploration. Because of space limitations we cannot present the complete CPN model which has been made available via [4].

2 Brief Overview of the DYMO Protocol

The operation of the DYMO protocol can be divided into two parts: *route discovery* and *route maintenance*. The route discovery part is used to establish routes between nodes in the network when required for communication between two nodes. A route discovery begins with an *originator node* multicasting a Route Request (RREQ) to all nodes in its immediate range. The RREQ has a *sequence number* to enable other nodes in the network to judge the freshness of the route request. The network is then flooded with RREQs until the request reaches its *target node* (provided that there exists a path from the originating node to the

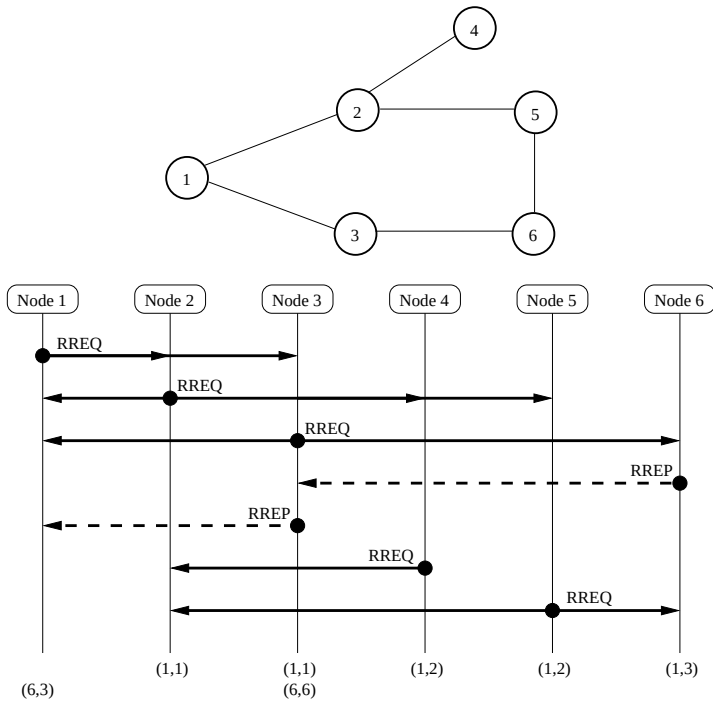


Fig. 1. DYMO route discovery example

target node). The target node then replies with a Route Reply (RREP) unicasted hop-by-hop back to the originating node. The route discovery procedure is requested by the IP network layer on a node when it receives an IP packet for transmission and does not have a route to the destination. An example scenario consisting of six nodes numbered 1–6 is shown in Fig. 1.

Figure 1(top) depicts the topology where an edge between two nodes indicates that the nodes are within direct transmission range of each other. As an example, node 1 is within transmission range of nodes 2 and 3. The message sequence chart (MSC) shown in Fig. 1 (bottom) depicts one possible exchange of messages in the DYMO protocol when the originating node 1 establishes a route to target node 6. Solid arcs represent multicast and dashed arcs represent unicast. In the MSC, node 1 multicasts a RREQ which is received by nodes 2 and 3. When receiving the RREQ from node 1, nodes 2 and 3 create an entry in their *routing table* specifying a route back to the originator node 1. Since nodes 2 and 3 are not the target of the RREQ they both multicast the received RREQ to their neighbours (nodes 1, 4 and 5, and nodes 1 and 6, respectively). Node 1 discards these messages as it was the originator of the RREQ. When nodes 4 and 5 receive the RREQ they add an entry to their routing table specifying that the originator node 1 can be reached via node 2. When node 6 receives the RREQ from node 3, it discovers that it is the target node of the RREQ, adds an entry to its routing

table specifying that node 1 can be reached via node 3, and unicasts a RREP back to node 3. When node 3 receives the RREP it adds an entry to its routing table stating that node 6 is within direct range, and use its entry in the routing table that was created when the RREQ was received to unicast the RREP to node 1. Upon receiving the RREP from node 3, node 1 adds an entry to its routing table specifying that node 6 can be reached using node 3 as a next hop. The RREQ is also multicasted by node 4, but when node 2 receives it it discards it as it will be considered *inferior*. Node 5 also multicasts the RREQ, but nodes 2 and 6 also consider the RREQ to be *inferior* and therefore discard the RREQ message. The two last lines in the MSC specifies the entries in the routing table of the individual nodes as a pair (*target*, *nexthop*). The first line specifies the entries that were created as a result of receiving the RREQ and the second line specifies entries created as a result of receiving the corresponding RREP. It can be seen that a bidirectional route has been discovered and established between node 1 and node 6 using node 3 as an intermediate hop.

The topology of a MANET network changes over time because of the mobility of the nodes. DYMO nodes therefore perform *route maintenance* where each node monitors the links to the nodes it is directly connected to. The DYMO protocol has a mechanism to notify nodes about a broken route. This is done by sending a Route Error (RERR), thereby informing nodes using the broken route that a new route discovery is needed.

The CPN model presented models the mandatory parts in revision 11 of the DYMO specification except for the modelling of actions to be taken if a node loses its sequence number, e.g., as a result of a reboot or crash ([2], Sect. 5.1.4). As we have decided to focus on the mandatory parts, our CPN model does not include the following optional DYMO operations: intermediate DYMO Router RREP creation ([2], Sect. 5.3.3), adding additional routing information to routing messages ([2], Sect. 5.3.5) and Internet attachment and gatewaying ([2], Sect. 5.8).

3 The DYMO CPN Model

The CPN model is a hierarchical model organised in 14 modules. Figure 2 shows the *module hierarchy* of the CPN model. Each node in Fig. 2 corresponds to a *module* and **System** represents the top-level module of the CPN model. An arc leading from one module to another module means that the latter module is a *submodule* of the former module. The model has been organised into two main parts: a **DYMOProtocol** part and a **MobileWirelessNetwork** part. This has been done to separate the parts of the model that are specific to DYMO which are all submodules of the **DYMOProtocol** module and the parts that are independent from the DYMO protocol which are all submodules of the **MobileWirelessNetwork** module. This means that the parts modelling the mobile wireless network over which DYMO operates can be reused for modelling other MANET protocols. We have adopted the convention that a *substitution transition* and its associated submodule have the same name. Furthermore, to the extent possible we have

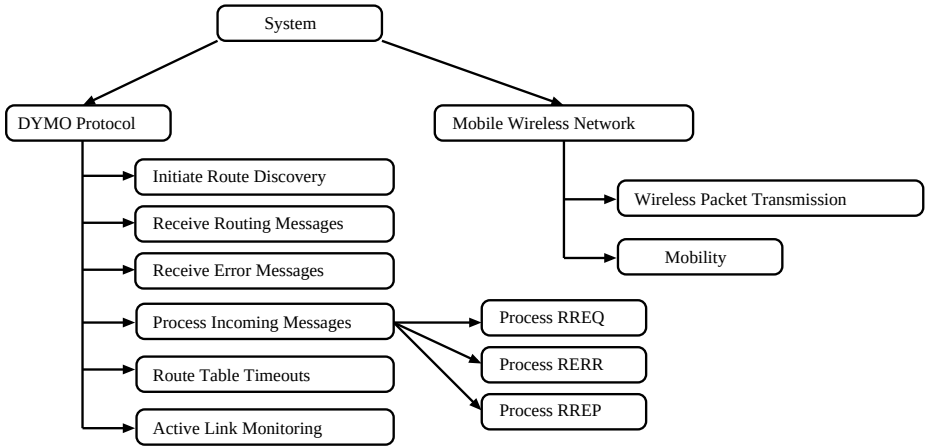


Fig. 2. Module hierarchy for the DYMO CPN model

structured the CPN model into modules such that it reflects the structure of the DYMO specification [2]. This makes the relationship between the DYMO specification and the CPN model more explicit, and it makes it easier to maintain the CPN model as the DYMO specification is being revised.

The top-level module **System** is shown in Fig. 3 and is used to connect the two main parts of the model. The DYMO protocol logic is modelled in the submodules of the **DYMOProtocol** substitution transition. The submodules of the **MobileWirelessNetwork** substitution transition is an abstract model of the mobile wireless network over which DYMO operates. It models unreliable one-hop wireless transmission of network packets over a network with mobile nodes. It models the operation of the IP network layer down to the physical transmission over the wireless medium.

The two socket places **DYMOToNetwork** and **NetworkToDYMO** are used to model the interaction between the DYMO protocol and the underlying protocol layers as represented by the submodules of the **MobileWirelessNetwork** substitution transition. The place **LinkState** is used to model the active link monitoring that nodes perform to check which neighbour nodes are still reachable. When the DYMO protocol module sends a message, it will appear as a token representing a network packet on place **DYMOToNetwork**. Similarly, a network packet to be received by the DYMO protocol module will appear as a token on the **NetworkToDYMO** place. The colour set **NetworkPacket** is defined as follows:

```

colset Node          = int with 0 .. N;
colset IPAddr        = union UNICAST : Node + LL_MANET_ROUTERS;
colset NetworkPacket = record src  : IPAddr * dest  : IPAddr *
                           data  : DYMOMessage;
  
```

We have used a record colour set for representing the network packets transmitted over the mobile wireless network. A network packet consists of a source,

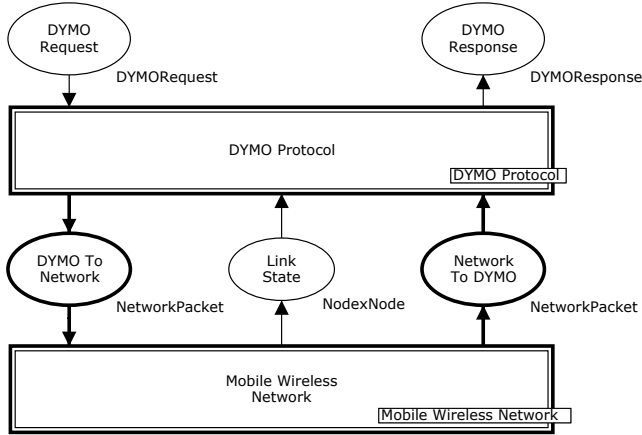


Fig. 3. Top-level System module of the CPN model

a destination, and some data (payload). The DYMO messages are carried in the data part and will be presented later. DYMO messages are designed to be carried in UDP datagrams transmitted over IP networks. This means that our network packets are abstract representations of IP/UDP datagrams. We have abstracted from all fields in the IP and UDP datagrams (except source and destination fields) as these do not impact the DYMO protocol logic. The source and destination of a network packet are modelled by the `IPAddr` colour set. There are two kinds of IP addresses: `UNICAST` addresses and the `LLMANET_ROUTERS` multicast address. The multicast address is used, e.g., in route discovery when a node is sending a `RREQ` to all its neighbouring nodes. Unicast addresses are used as source of network packets and, e.g., as destinations in `RREP` messages. A unicast address is represented using an integer from the colour set `Node`. Hence, the model abstracts from real IP addresses and identify nodes using integers in the interval $[1; N]$ where N is a model parameter specifying the number of nodes in the MANET.

The two places `DYMORequest` and `DYMOResponse` are used to interact with the service provided by the DYMO protocol. A route discovery for a specific destination is requested via the `DYMORequest` place and a DYMO response to a route discovery is then provided by DYMO via the `DYMOResponse` place. The colour sets `DYMORequest` and `DYMOResponse` are defined as follows:

```

colset RouteRequest = record originator : Node * target : Node;
colset DYMORequest  = union ROUTEREQUEST : RouteRequest;

colset RouteResponse = record originator : Node * target : Node *
                           status       : BOOL;
colset DYMOResponse  = union ROUTERESPONSE : RouteResponse;

```

A `DYMORequest` specifies the identity of the originator node requesting the route and the identity of the target node to which a route is to be discovered. Similarly,

a `DYMOResponse` contains a specification of the originator, the target, and a boolean status specifying whether the route discovery was successful. The colour sets `DYMORequest` and `DYMOResponse` are defined as union types to make it easy to later extend the model with additional requests and responses. This will be needed when we later refine the CPN model to more explicitly specify the interaction between the DYMO protocol module and the IP network layer module. By setting the initial marking of the place `DYMORequest`, it can be controlled which route requests are to be made. We have not modelled the actual transmission of messages containing payload from applications as our focus is on the route establishment and maintenance of the DYMO protocol. In the following we present the submodules of the `DYMOProtocol` and `MobileWirelessNetwork` substitution transitions in more detail.

3.1 The DYMO Protocol Module

The top-level module for the DYMO protocol part of the CPN model is the `DYMOProtocol` module shown in Fig. 4. The module has five substitution transitions modelling the handling of route request from a user (`InitiateRouteDiscovery`), reception of routing messages (`ReceiveRoutingMessages`) which are `RREQ` and `RREP` messages, the reception of `RERRs` (`ReceiveErrorMessages`), processing of incoming messages (`ProcessIncomingMessages`), and timer management associated with the routing table entries (`RouteTableEntryTimeouts`). All submodules of the substitution transitions in Fig. 4 create and manipulate DYMO messages which are represented by the colour set `DYMOMessage` defined as follows:

```
colset SeqNum = int with 0 .. 65535;

colset NodexSeqNum      = product Node * SeqNum;
colset NodexSeqNumList = list NodexSeqNum;

colset RERRMessage = record HopLimit      : INT *
                          UnreachableNodes : NodexSeqNumList;

colset RoutingMessage = record TargetAddr : Node   * OrigAddr : Nodes *
                          OrigSeqNum : SeqNum * HopLimit : INT   *
                          Dist       : INT;

colset DYMOMessage = union RREQ : RoutingMessage + RREP : RoutingMessage +
                          RERR : RERRMessage;
```

The definition of the colour sets used for modelling the DYMO messages is based on a direct translation of the description of DYMO messages as found in Sect. 4.2.2 and Sect. 4.2.3 of the DYMO specification [2]. In particular we use the same names of message fields as in [2]. In the modelling of DYMO message packets, we have abstracted from the specific layout as specified by the `packetbb` format [8]. This is done to ease the readability of the CPN model, and the packet layout is not important when considering only the functional operation of the DYMO protocol.

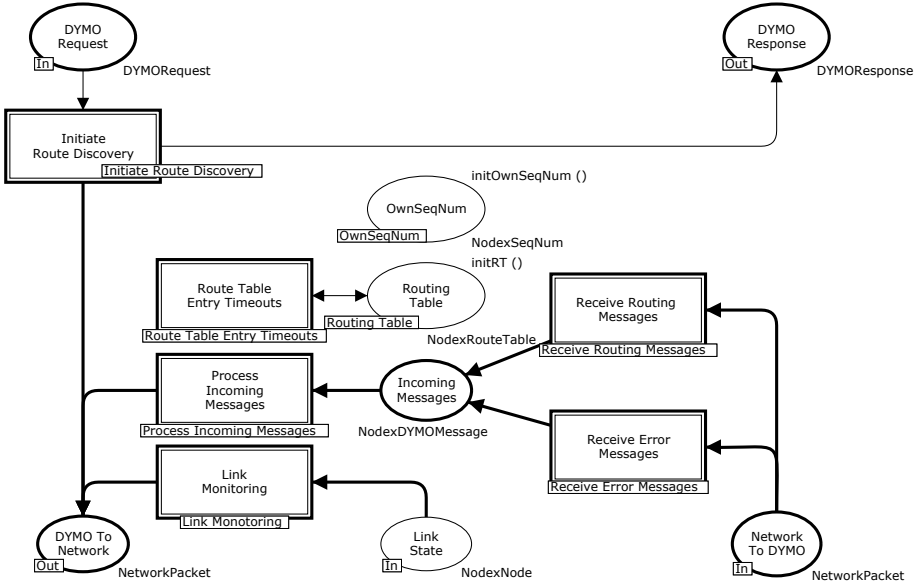


Fig. 4. The DYMOProtocol module

The submodules of the DYMOProtocol module also access the routing table and the sequence number maintained by each mobile node. To reflect this and reduce the number of arcs in the modules, we decided to represent the routing table and the node sequence numbers using fusion places. The place **OwnSeqNum** contains a token for each node specifying the current sequence number of the node and the place **RoutingTable** contains a token for each node specifying a routing table. The colour set **SeqNum** used to represent the unsigned 16-bit sequence number of a node was defined above and the colour set **RouteTable** used to represent the routing table of a node is defined as follows:

```
colset RouteTableEntry = record
    Address      : IPAddr * SeqNum : SeqNum *
    NextHopAddress : IPAddr * Broken : BOOL   *
    Dist         : INT;

colset RouteTable      = list RouteTableEntry;
colset NodexRouteTable = product Node * RouteTable;
```

A routing table is represented as a list of **RouteTableEntry**. To allow each node to have its own routing table, we use the colour set **NodexRouteTable** for representing the set of routing tables. The first component of a pair specifies the node to which the routing table in the second component is associated. The definition of the colour **RouteTableEntry** is a direct translation of routing table entries as described in Sect. 4.1 of [2]. In addition to the mandatory fields, we

have included the optional `Dist` field as this is used in non-trivial ways in the reception of routing messages. We wanted to investigate this operation in more detail as part of the state space exploration and we have therefore included it in the modelling.

Initiate Route Discovery Module. Figure 5 shows the `InitiateRouteDiscovery` module. When a route request arrives via the `DYMOREquest` input port the `ProcessRouteRequest` transition is enabled. When it occurs it will initialise the processing of the route request by putting a token on place `Processing`. A route request being processed is represented by a token over the colour set `NodeRCxRouteRequest` which is a product type where the first component specifies the node processing the route request (i.e., the originator), the second component specifies how many times the RREQ has been retransmitted, and the third component specifies the route request. If the node does not have a route to the target and the retransmit limit `RREQ_TRIESReached` for RREQs has not been reached (as specified by the guard of the `CreateRREQ` transition), then a RREQ message can be transmitted with the current sequence number of the node. Upon sending a RREQ, the sequence number of the node is incremented and so is the counter specifying how many times the RREQ has been transmitted. If a route becomes established (i.e., the originator receives a RREP on the RREQ), the `RouteEstablished` transition becomes enabled and a token can be put on place `DYMOREsponse` indicating that the requested route has been successfully established. If the retransmission limit for RREQs is reached (before a RREP is received), the `RREQ_TRIES` transition becomes enabled and a token can be put on place `DYMOREsponse` indicating that the requested route could not be established.

Receive Routing Messages Module. When a routing message arrives at the DYMO protocol module on the place `NetworkToDYMO`, the first task is to compare the routing information in the received routing message with the information contained in the routing table of the receiving node. Judging the routing information contained in a message is handled by the `ReceiveRoutingMessages` module shown in Fig. 6. The receiver is found in the `dest` field of the incoming network packet bound to `np`. This way we know which node is the current node and thereby which routing table to access. Section 5.2.1 of [2] specifies how routing information is divided into four classes. Each class has a boolean expression specifying when routing information falls into the given class. Each class is represented by an accordingly named transition with a guard which is true exactly when the boolean expression corresponding to that class is true.

The first class `Stale` is routing information which is considered outdated and hence not suitable for updating the routing table. The function `isStale` is given a network packet `np` and the routing table `rt` and returns true if and only if the routing information in the packet is stale. The second class is `LoopPossible` which is routing information where using it to update the routing table may introduce routing loops. Routing information falls into the class `Inferior` if we already have a better route to the node. The last of the four classes is `Superior`

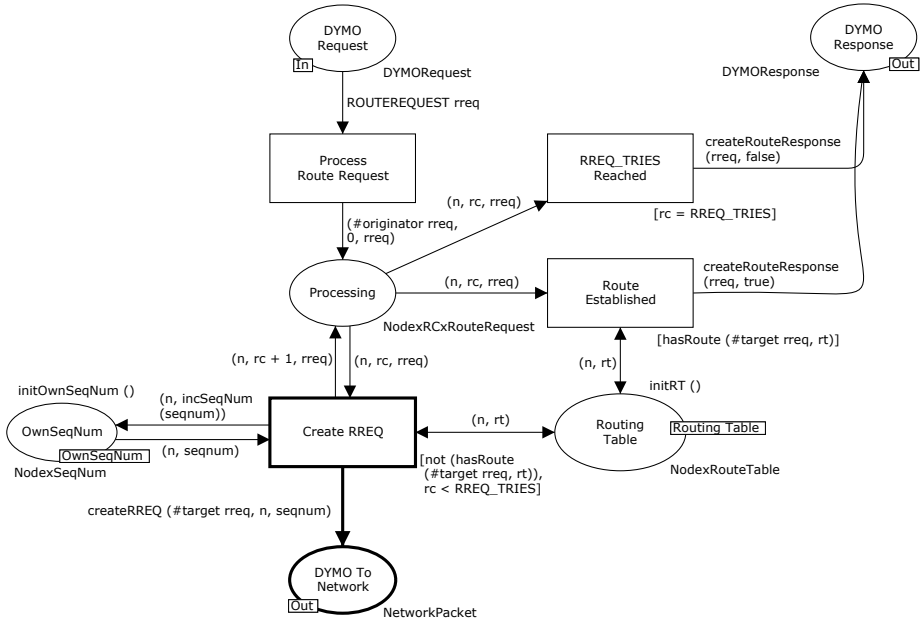


Fig. 5. The Initiate Route Discovery module

routing information. This is routing information which is considered better than the information present in the routing table and is therefore used to update the entry to the originating node using the function `updateRouteEntry`.

If there is no entry to the originating node, the transition `NewRoute` is enabled and when it occurs a new entry is made by the function `newRouteEntry` which conforms to Sect. 5.2.2 of [2]. Network packets originating from the current node are discarded. Altogether this results in only network packets with superior or new routing information being passed from the `ReceiveRoutingMessage` module to the place `IncomingMessages` for further processing.

Process RREQ. As a representative example of a submodule at the most detailed level of the CPN model, we consider the `ProcessRREQ` module shown in Fig. 7 which specifies the processing of RREQ messages. The submodules specifying the processing of RREP and RERR are similar.

There are two cases in processing a RREQ: either the receiving node is the target for the RREQ or not. If the node is not the target node, the transition `RREQForward` is enabled. The function `forwardRREQ` placed on the outgoing arc conforms to Sect. 5.3.4 of [2], and works in the following way. If the `HopLimit` is greater than or equal to one, the RREQ message has to be forwarded. This is done by creating a new network packet with `dest` set to the `LL_MANET_ROUTERS` multicast address and `src` set to the current node. The `TargetAddr`, `OrigAddr`, and `OrigSeqNum` in the message is not changed, but `HopLimit` is decreased by

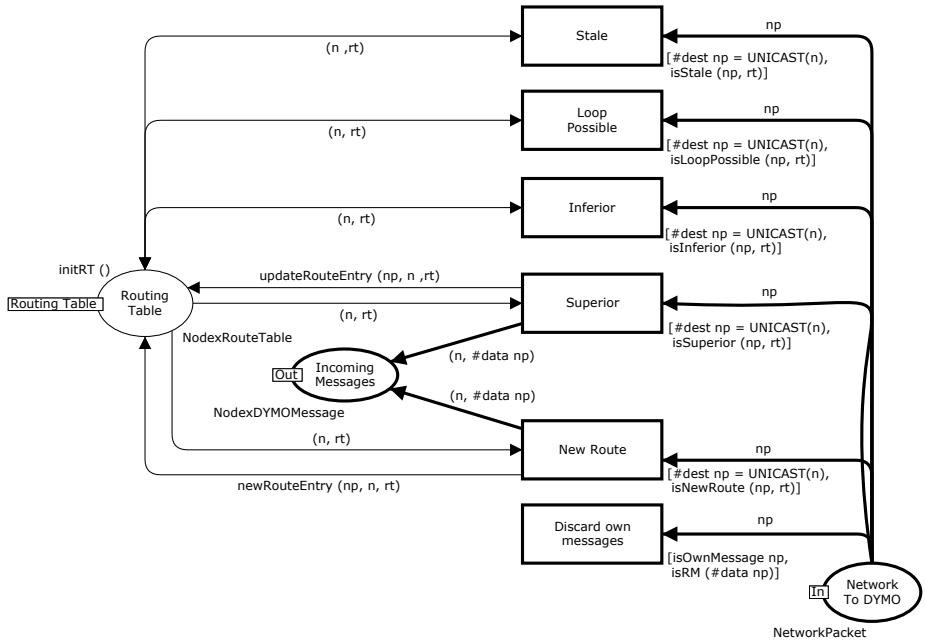


Fig. 6. The Receive Routing Message module

one and `Dist` is increased by one. If the `HopLimit` is one, the function simply returns the empty multi-set, i.e., the message is discarded.

If the current node is the target of the request the transition `RREQTarget` is enabled. The function `createRREP` creates a `RREP` message where the `dest` field is set to the `nextHopAddress` for the originating node given in the routing table. The `src` is set to the current node, `TargetAddr` is set to the originator of the `RREQ`, and `OrigAddr` is set to the current node.

3.2 Mobile Wireless Network

The `MobileWirelessNetwork` module shown in Fig. 8 is an abstract representation of the MANET that DYMO is designed to operate over. It consists of two parts: a part modelling the transmission of network packets represented by the substitution transition `WirelessPacketTransmission`, and a part modelling the mobility of the nodes represented by the `Mobility` substitution transition. The transmission of network packets is done relative to the current topology of the MANET which are explicitly represented via the current marking of the `Topology` place. The topology is represented using the colour set `Topology` defined as follows:

```

colset NodeList = list Node;
colset Topology = product Node * NodeList;

```

The idea is that each node has an adjacency list of nodes that it can reach in one hop, i.e., its neighbouring nodes. This adjacency list is then consulted when

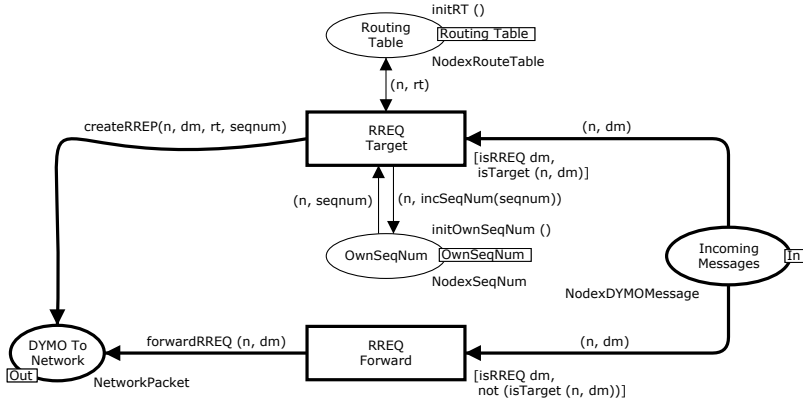


Fig. 7. The ProcessRREQ module

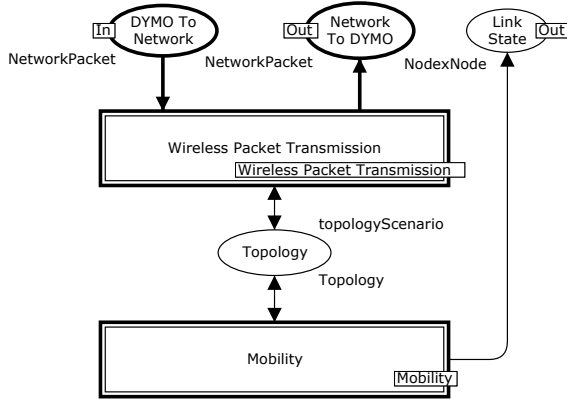


Fig. 8. The Mobile Wireless Network

a network packet is being transmitted from a node to determine the set of nodes that can receive the network packet. In this way, we can model a mobile network where the topology is dynamic by simply adding or removing nodes from the adjacency lists. The place **LinkState** models that a node can be informed about reachability of its neighbouring nodes which is used in active link monitoring.

The **WirelessPacketTransmission** module models the actual transmission of packets and is shown in Fig. 9. In this module, it is modelled how the network packets are transmitted via the physical network. Packets are transmitted over the network according to the function **transmit** on the arc from the transition **Transmit** to the place **NetworkToDYMO**. The **transmit** function starts out by checking if the **success** argument is true and if not, the packet is discarded. This corresponds to modelling a simple unreliable network. If the packet is to be successfully sent and the destination address is the multicast address, the

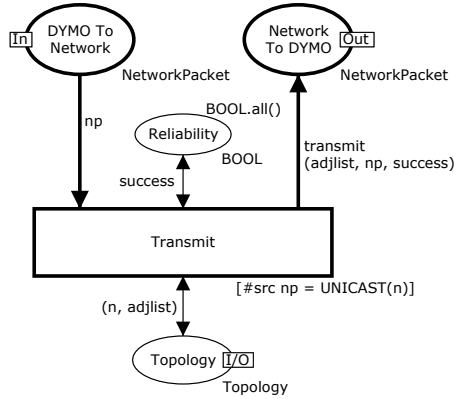


Fig. 9. The Wireless Packet Transmission

packet is sent to each of the nodes in the adjacency list of the transmitting node. If the destination address is a unicast address and the address exists in the adjacency list of the transmitting node, i.e., the destination node is within range, then the packet is forwarded. It should be noted that in reality a transmission could be received by any subset of the neighbouring nodes, e.g., because of signal interference. We only model that either all of the neighbouring nodes receives the packet or none receives it. This is sufficient because our modelling of the dynamic topology means that a node can move out of reach of the transmitting node immediately before the transmission occurs which has exactly the same effect as a signal interference in that the node does not receive the packet. Hence, signal interference and similar phenomena implying that a node does not receive a packet is in our model equivalent to the node moving out of reach of the transmitting node.

The dynamic changes in the topology of the MANET is modelled in the **Mobility** module (not shown). The basic idea in the modelling of the mobility is to explicitly represent the mobility scenario considered, i.e., the sequences of link changes that are possible in the scenario. The reason for having this explicit topology control is that in a MANET with N nodes and symmetric links there are $\frac{N*(N-1)}{2}$ possible links between the nodes and therefore $2^{\frac{N*(N-1)}{2}}$ possible topologies. Hence, this will make state space analysis impossible for dynamic topologies because of state explosion. By having explicit topology control in the model, we can limit the number of possible combinations and consider different mobility scenarios one at a time. The model can capture a fully dynamic topology by specifying a mobility scenario where any link can go up or down, and it can also capture the static scenario with no topology changes.

3.3 Results from Modelling

Our modelling work started when version 10 [3] was the most recent DYMO specification. In the process of constructing the CPN model and simulating it,

we have discovered several issues and ambiguities in the specification. The most important ones were:

1. When processing a routing message, a DYMO router may respond with a RREQ flood, i.e., a RREQ addressed to the node itself, when it is target for a RREQ message (cf. [3], Sect. 5.3.4). It was not clear which information to put in the RREQ message, i.e., the originator address, hop limit, and sequence number of the RREQ.
2. When judging the usefulness of routing information, the target node is not considered. This means that a new request with a higher sequence number can make an older request for another node stale since the sequence number in the old message is smaller than the sequence number found in the routing table.
3. When creating a RREQ message the distance field in the message is set to zero. This means that for a given node n an entry in the routing table of a node n' connected directly to n may have a distance to n which is 0. Distance is a metric indicating the distance traversed before reaching n , and the distance between two directly connected nodes should be one.
4. In the description of the data structure route table entry (cf. [3], Sect. 4.1) it is suggested that the address field can contain more than one node. It was not clear why this is the case.
5. When processing RERR messages (cf. [3], Sect. 5.5.4) it is not specified whether hop limit shall be decremented.
6. When retransmitting a RREQ message (cf. [3], Sect. 5.4), it was not explicitly stated whether the node sequence number is increased.
7. Version 10 of DYMO introduced the concept of distance instead of hop count. The idea is that distance is a more general metric, but in the routing message processing (cf. [2], Sect. 5.3.4) it is incremented by one. We believe it should be up to the implementers how much distance is incremented depending on the metric used.

These issues were submitted to the IETF MANET Working Group mailing list [12] and issue 1 and 3-7 were acknowledged by the DYMO developers and version 11 of the DYMO specification [2] has been modified to resolve the issues. Issue 2 still remains to be resolved, but it is not truly critical as it only causes route discovery to fail in scenarios which according to the experience of the DYMO developers seldom occur in practice.

4 Initial State Space Exploration

In this section we present our initial state space exploration conducted on the DYMO protocol model in a number of scenarios using the state space tool of

CPN Tools [5]. A scenario is defined by specifying the route discoveries to be made, the initial topology, and the possible topology changes. In the initial analysis we consider only scenarios with a static topology and with symmetric links. Furthermore, we consider a reliable network, i.e., a network that cannot lose network packets and we consider only scenarios with one route request. These scenarios allows us to generate state spaces of a reasonable size.

When considering which scenarios to investigate, we observed that some scenarios can be considered equivalent (symmetric). As an example consider the example scenario depicted in Fig. 10 (left). The figure shows a topology with three nodes where node 1 is connected to node 2 via a symmetric link and similar for node 2 and 3. The arrow represents a request for a route discovery where the source node of the arrow is the originator node of the request and the destination node of the arrow is the target node of the request. In this case node 1 is requesting a route to node 3. But if we permute the identity of node 1 and 3, we have exactly the same scenario as in Fig. 10 (right). We will call such scenarios equivalent and only explore one representative scenario from each such equivalence class. It is important to notice that equivalence is both with respect to topology and the originator and target of route requests. Hence, two scenarios can be considered equivalent if one can be obtained from the other by a permutation of node identities. In this way, we can reduce the number of scenarios that needs to be considered.

For scenarios containing two nodes we only have a single equivalence class. Looking at scenarios with three nodes we have four equivalence classes, and with four nodes we have 19 equivalence classes. In the following we consider representatives for each equivalence class in scenarios with two and three nodes. For each representative scenario, we also explore the state space when `RREQ_TRIES` is increased from 1 to 2. Additionally, we look at two representative scenarios for equivalence classes of scenarios with four nodes. All scenarios were analysed using state space exploration by first generating the full state space, then generating the Strongly Connected Component Graph (SCC-graph), and finally generating the state space report. The analysis focuses on the dead markings which represents states where the protocol has terminated.

In addition to considering the dead markings, we also investigated the properties of the protocol with respect to classifying the incoming routing information against the information in the routing table. Just by looking at the boolean expressions for the four classes (inferior, superior, loop possible, and stale) in the DYMO specification it is hard to tell if it is always the case that new routing information only falls into one of the classes. In the model this corresponds to the enabling of the relevant transition in Fig. 6 being *exclusive* in the sense

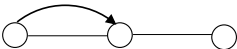
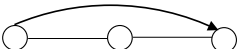
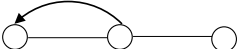
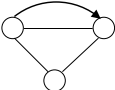
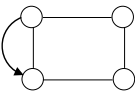
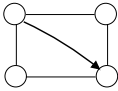


Fig. 10. Example scenario with three nodes and one route request

that only one of them is enabled for a given incoming network packet. Using the state space, we investigated if there are cases where routing information falls into more than one class. For this purpose we implemented a state predicate **CheckExclusive** which is true in a given state if more than one of the five transitions for judging routing information in Fig. 6 were enabled for a given incoming packet.

Table 1 summaries the state space results for the scenarios considered. In the first column is a graphical representation of the scenario considered. The second column shows the value of **RREQ_TRIES**, and the third and fourth column list the number of nodes and arcs in the state space, respectively. The last column shows the dead markings found in the state space. In the first four scenarios, we can see that with **RREQ_TRIES** set to 1 we get two dead markings. The first dead marking is the state where **RREQ_TRIESReached** (see Fig. 5) has occurred before a **RREP** was received and the routing table updated. The second dead marking is the state where **RREQ_TRIESReached** did not occur and **RouteEstablished** occurred after the routing table had been updated. By transferring the dead marking into the simulator, we inspected the marking and observed that these are desired terminal states of the protocol, i.e., states where the requested route was established. The five dead markings we get when **RREQ_TRIES** is set to 2 is caused by the same sequence of occurrences as above, but here we also have overtaking on the network causing routing information to become stale and therefore a different sequence number is put in the routing table which results in a different marking.

Table 1. Summary of state space exploration results

Scenario	RREQ TRIES	Nodes	Arcs	Dead markings
	1	18	24	[17,18]
	2	145	307	[50,118,119,142,143]
	1	18	24	[17,18]
	2	145	307	[50,118,119,144,145]
	1	50	90	[49,50]
	2	1,260	3,875	[372,704,705,1219,1220]
	1	74	156	[73,74]
	2	2,785	10,203	[868,2435,2436,2758,2759]
	1	446	1,172	[444,443,404,403,283,282]
	2	166,411	804,394	[98796,98795,9856,97625,...] (23)
	1	1,098	3,444	[852,851,551,550,1096,1095]
	1	558	1,606	[555,556,557,558]

Common to all scenarios we have listed in Tab. 1 is that manual inspection of the dead markings showed that the model had reached a state where the requested route had actually been established. By a route being established we mean that the originator of a request has a route entry to the target node of the request and if we follow the `NextHopAddress` hop-by-hop we can get from the originating node to the target node and vice versa. Another common result for all scenarios is that the SCC graph has the same number of nodes and arcs as the state space graph. This means that there are only trivial SSCs and this implies that there are no cycles in the state space and the protocol will therefore always terminate.

Evaluating the function `CheckExclusive` on the CPN model corresponding to version 10 of DYMO revealed that in the second last scenario in Tab. 1 the judging of routing information was not exclusive as it was possible for a routing message to be judged as both loop-possible and inferior. Since network packets containing such routing information are discarded this is not a major problem. This issue was submitted to the MANET Working Group as part of the issues discussed in Sect. 3.3. This prompted a modification of the boolean expressions for classifying routing information in version 11 of DYMO and rerunning our analysis with the CPN model for version 11 showed that the issue had been resolved.

5 Conclusion and Future Work

We have investigated the DYMO routing protocol and demonstrated how the construction of executable formal models (such as a CPN model) can be a very effective way of systematically reviewing an industrial-size protocol specification. The DYMO protocol specification can be considered a fairly mature specification. Even so, our modelling and validation work revealed several cases of ambiguity and incompleteness, which were discovered through both modelling, simulation and state space exploration. Our findings were subsequently provided to DYMO developers and thereby contributed to the development of the DYMO protocol. Altogether approximately 300 person-hours were used on the modelling and validation work presented in this paper.

Modelling and validation of routing protocols for MANETs has also been considered by other researchers. The AODV protocol was considered in [17,18], the DSDV protocol was presented in [19], and the WARP Routing Protocol was modelled and verified in [6]. The LUNAR routing protocol was modelled and verified in [13], and the RIP and AODV protocols were verified in [1] using a combination of model checking and theorem proving. Closest to our work is the CPN modelling of the DYMO protocol presented in [20] which considered the 5th revision of the DYMO specification. A main difference between our modelling approach and [20] is that [20] presents a highly compact CPN model of the DYMO protocol consisting of just a single module. Furthermore, [20] uses simulation to investigate properties of the protocol whereas we presents some initial state space analysis results. Our aim is to eventually use the CPN model as a basis for implementing the DYMO protocol. We therefore decided on a

more verbose modelling approach which includes organising the CPN model into several modules to break up the complexity of modelling the complete DYMO protocol, and make it easier to later refine parts of the CPN model as required. The work of [20] appears to aim at state space analysis, which is why it is beneficial to have a compact CPN model to reduce the effect of state explosion. This is an often encountered trade-off between state space size and compactness of the CPN model. On the other hand, [20] also models the optional appending of information to the routing messages and report on several problematic issues in the DYMO specification. Compared to [20], we have the current topology explicitly represented in the marking and have explicit control of the possible topology changes that can occur. The main motivation for this is to make a scenario based state space analysis possible as illustrated in this paper. The approach of [20] relies on an abstract and implicit modelling of the topology where a transmitted network packet is either received by a single node or by no nodes. As stated in [20], this should be extended such that any number of nodes can receive a transmitted message. It remains to be investigated which of the two approaches to modelling dynamic topology is most suited in terms of state space size and properties preserved.

We plan to continue our work by extending the state space exploration results to cover more scenarios and properties. This will include the application of state space reduction methods and exploiting the symmetry in the scenarios to reduce the number of cases that needs to be considered. The next step in our modelling work is to refine the CPN model such that the interaction between the DYMO protocol and the surrounding IP network layer becomes explicit. This is required in order to use the CPN model as a basis for implementing the DYMO routing protocol daemon which is the long term goal of the project.

Acknowledgements. The authors wish to thank the anonymous reviewers for their constructive comments and suggestions that have helped us to improve the CPN model and paper.

References

1. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal Verification of Standards for Distance Vector Routing Protocols. *Journal of the ACM* 49(4), 538–576 (2002)
2. Chakeres, I.D., Perkins, C.E.: Dynamic MANET On-demand (DYMO) Routing. Internet-Draft. Work in Progress (November 2007), <http://www.ietf.org/internet-drafts/draft-ietf-manet-dymo-11.txt>
3. Chakeres, I.D., Perkins, C.E.: Dynamic MANET On-demand (DYMO) Routing. Internet-Draft. Work in Progress (July 2007), <http://www.ietf.org/internet-drafts/draft-ietf-manet-dymo-10.txt>
4. CPN DYMO Model, <http://www.daimi.au.dk/~kris/cpndymo.cpn>
5. CPN Tools Home page, <http://www.daimi.au.dk/CPNTools>
6. de Renesse, R., Aghvami, A.H.: Formal Verification of Ad-Hoc Routing Protocols using SPIN Model Checker. In: *Proc. of IEEE MELECON*, pp. 1177–1182 (2005)

7. Espensen, K.L., Kjeldsen, M.K., Kristensen, L.M.: Towards Modelling and Verification of the DYMO Routing Protocol for Mobile Ad-hoc Networks. In: Proc. of CPN 2007, pp. 243–262 (2007)
8. Clausen, T., et al.: Generalized MANET Packet/Message Format. Internet-draft (2007) (work in progress)
9. Hansen, S.: Modelling and Validation of the Dynamic On-Demand Routing (DYMO) Protocol. Master's thesis, Department of Computer Science, University of Aarhus (in danish) (February 2007)
10. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer (STTT) 9(3-4), 213–254 (2007)
11. IETF MANET Working Group,
<http://www.ietf.org/html.charters/manet-charter.html>
12. IETF Mobile Ad-hoc Networks Discussion Archive,
<http://www1.ietf.org/mail-archive/web/manet/current/index.html>
13. Wibling, O., Parrow, J., Pears, A.: Automatized Verification of Ad Hoc Routing Protocols. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 343–358. Springer, Heidelberg (2004)
14. Perkins, C.E.: Ad Hoc Networking. Addison-Wesley, Reading (2001)
15. Thorup, R.: Implementation and Evaluation of the Dynamic On-Demand Routing (DYMO) Protocol. Master's thesis, Department of Computer Science, University of Aarhus (February 2007)
16. Thouvenin, R.: Implementation of the Dynamic MANET On-Demand Routing Protocol on the TinyOS Platform. Master's thesis, Department of Computer Science, University of Aarhus (July 2007)
17. Xiong, C., Murata, T., Leigh, J.: An Approach to Verifying Routing Protocols in Mobile Ad Hoc Networks Using Petri Nets. In: Proceedings. of IEEE 6th CAS Symposium on Emerging Technologies, pp. 537–540 (2004)
18. Xiong, C., Murata, T., Tsai, J.: Modeling and Simulation of Routing Protocol for Mobile Ad Hoc networks Using Colored Petri Nets. Research and Practice in Information Technology 12, 145–153 (2002)
19. Yuan, C., Billington, J.: An Abstract Model of Routing in Mobile Ad Hoc Networks. In: Proc. of CPN 2005, pp. 137–156. DAIMI PB-576 (2005)
20. Yuan, C., Billington, J.: A Coloured Petri Net Model of the Dynamic MANET On-demand Routing Protocol. In: Proc. of Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pp. 37–56 (2006)

Formal Specification and Validation of Secure Connection Establishment in a Generic Access Network Scenario^{*}

Paul Fleischer and Lars M. Kristensen

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
{pf,kris}@daimi.au.dk

Abstract. The Generic Access Network (GAN) architecture is defined by the 3rd Generation Partnership Project (3GPP), and allows telephone services, such as SMS and voice-calls, to be accessed via generic IP networks. The main usage of this is to allow mobile phones to use WiFi in addition to the usual GSM network. The GAN specification relies on the Internet Protocol Security layer (IPSec) and the Internet Key Exchange protocol (IKEv2) to provide encryption across IP networks, and thus avoid compromising the security of the telephone networks. The detailed usage of these two Internet protocols (IPSec and IKEv2) is only roughly sketched in the GAN specification. As part of the process to develop solutions to support the GAN architecture, TietoEnator Denmark has developed a detailed GAN scenario which describes how IPsec and IKEv2 are to be used during the connection establishment procedure. This paper presents a CPN model developed to formally specify and validate the detailed GAN scenario considered by TietoEnator.

1 Introduction

The Generic Access Network (GAN) [1] architecture specified by the 3rd Generation Partnership Project (3GPP) [2] allows access to common telephone services such as SMS and voice-calls via generic Internet Protocol (IP) networks. The operation of GAN is based on a *Mobile Station* (e.g., a cellular phone) opening an encrypted tunnel to a *Security Gateway* via an IP network. A *GAN Controller* is responsible for relaying the commands sent via this tunnel to the telephone network, which in turn allows mobile stations to access the services on the telephone network. The Security Gateway and the GAN Controller can either reside on the same physical machine or on two separate machines. The encrypted tunnel is provided by the Encapsulating Security Payload (ESP) mode of the IP security layer (IPSec) [14]. To provide such an encrypted tunnel, both ends have to authenticate each other, and agree on both encryption algorithm and keys. This is achieved using the Internet Key Exchange v2 (IKEv2) protocol [7]. The

^{*} Supported by the Danish Research Council for Technology and Production and TietoEnator Denmark.

GAN specification [1] merely states that IKEv2 and IPSec are to be used, and in which operating modes. However, what that means for the message exchange is not specified, and is left to the IKEv2 and IPSec standards. As such, there is no clear specification of the IKEv2 message exchange and the states that the protocol entities are to be in when establishing a GAN connection.

TietoEnator Denmark [5] is working on providing solutions to support the GAN architecture. Prior to the implementation, a textual usage scenario was formulated [11] which constitutes a specific instantiation of the full GAN architecture. The purpose of this scenario was two-fold. First, it defines the scope of the software to be developed, i.e., which parts of the full GAN specification are to be supported. Secondly, the scenario describes thoughts about the initial design of both the software and the usage of it. The scenario describes the details of how a mobile station is configured with an IP address using DHCP [6] and then establishes an ESP-tunnel [13] to the Security Gateway using IKEv2 [7] and IPSec [14]. At this point, the mobile station is ready to communicate securely with the GAN Controller. The focus of the scenario is the establishment of the secure tunnel and initial GAN message exchanges which are the detailed parts omitted in the full GAN specification. Throughout this paper the term *GAN scenario* refers to the detailed scenario [11] described by TietoEnator, while *GAN architecture* refers to the generic architecture as specified in [1].

The contribution of this paper is to describe the results of an industrial project at TietoEnator, where Coloured Petri Nets (CPNs) [12] were used as a supplement to a textual description of the GAN scenario to be implemented. The model has been constructed from the material available from TietoEnator [11], the GAN specification [1], and the IKEv2 specification [7]. The CPN model deals only with the connection establishing aspect of the GAN architecture, as this is the main focus of the TietoEnator project. As the scenario from TietoEnator deals with the aspect of configuring the mobile station with an initial IP address, the model does not only cover the communication of the GAN protocol, but also of the DHCP messages and actual IP-packet flow. The CPN model includes a generic IP-stack model, which supports packet forwarding and ESP-tunnelling. This modelling approach was chosen to allow the model to be very close to the scenario description used by TietoEnator, with the aim of easing the understanding of the model for TietoEnator engineers which will eventually implement the GAN scenario. The model was presented to the engineers at two meetings. Each meeting resulted in minor changes of the model. Even though the TietoEnator engineers did not have any previous experience with CPNs, they quickly accepted the level of abstraction and agreed that the model reflected the scenario they had described in the textual specification.

Coloured Petri Nets have been widely used to model and validate Internet protocols. Recent examples include the ERDP [15], TCP [3], and DCCP [16] protocols. The general approach of modelling Internet protocols is to abstract as much of the environment away, and only deal with the core operation of the protocol. The advantage of this approach is that the model gets simpler and the analysis becomes easier due to restricted state space size. The approach presented

in this paper also makes use of abstraction. However, the chosen abstraction level is based on a protocol architecture, rather than a single protocol specification. This gives a complete picture of the architecture, rather than a single protocol. This is an advantage when working with the design of actual protocol implementations as it gives an overview of the needed features and component interaction. In particular this means that our model is aimed at validating the interaction between several protocol components instead of a single protocol as done in, e.g., [3,15,16]. To the best of our knowledge, the GAN architecture has not previously been exposed to formal modelling. Preliminary reports on the project presented in this paper have appeared in [8,9].

This paper is organised as follows. Sect. 2 gives an introduction to the GAN scenario as defined by TietoEnator and presents the top-level of the constructed CPN model. Sect. 3 and Sect. 4 present selected parts of the constructed CPN model including a discussion of the modelling choices we made. In Sect. 5, we explain how the model of the GAN specification was validated using simulation and state space analysis. Finally, Sect. 6 sums up the conclusions and discusses future work. The reader is assumed to be familiar with the basic ideas of the CPN modelling language [12] as supported by CPN Tools [4].

2 The GAN Scenario

This section gives an introduction to the GAN scenario [11] as defined by TietoEnator and the constructed CPN model. Fig. 1 shows the top-level module of the constructed CPN model. The top-level module has been organised such that it reflects the network architecture of the GAN scenario. The six substitution transitions represent the six network nodes in the scenario and the four places with thick lines represent networks connected to the network nodes. The places with thin lines connected to the substitution transitions Provisioning Security Gateway, Default Security Gateway, Provisioning GAN Controller, and Default GAN Controller are used to provide configuration information to the corresponding

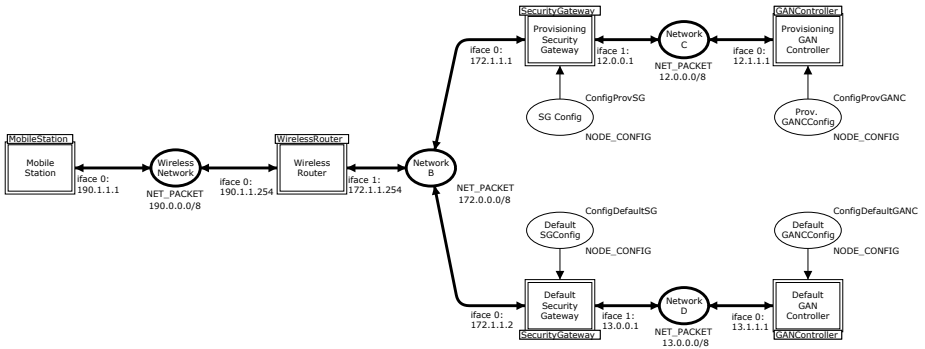


Fig. 1. Top-level module of the CPN model

network nodes. The module has been annotated with network and interface IP addresses to provide that information at the topmost level.

The substitution transition **Mobile Station** represents the mobile station which is connecting to the telephone network via a generic IP network. The place **Wireless Network** connected to **MobileStation** represents the wireless network which connects the mobile station to a wireless router represented by the substitution transition **Wireless Router**. The wireless router is an arbitrary access point with routing functionality, and is connected to the **ProvisioningSecurityGateway**, through **Network B**. The provisioning security gateway is connected to the **ProvisioningGANController** via **Network C**. There is a second pair of security gateway (**DefaultSecurityGateway**) and GAN controller (**DefaultGANController**). The provisioning GAN controller is responsible for authenticating any connection, and redirecting them to another GAN controller in order to balance the load over a number of GAN controllers. In the GAN scenario, the default GAN controller represents the GAN controller which everything is redirected to. It is worth mentioning, that the generic GAN architecture sees the security gateway as a component within the GAN controller. However, TietoEnator decided to separate the two to make the message exchange more clear. Neither the **Wireless Router** nor **Network B** are required to be owned or operated by the telephone operator. However, all security gateways, GAN controllers, and **Network C** and **Network D** are assumed to be under the control of the telephone operator, as non-encrypted messages will be exchanged across them.

The basic exchange of messages in the GAN scenario consists of a number of steps as depicted in the Message Sequence Charts (MSCs) in Figs. 2-4. The MSCs have been automatically generated from the constructed CPN model using the BritNeY tool [17]. The scenario assumes that the **Mobile Station** is completely off-line to begin with. It then goes through 5 steps: Acquire an IP address using DHCP, create a secure tunnel to the provisioning security gateway, use the GAN protocol to acquire the addresses of the security gateway and GAN controller to use for further communication, create a secure tunnel to the new security gateway, and finally initiate a GAN connection with the new GAN controller.

The first step is to create a connection to an IP network which is connected to the **Provisioning Security Gateway** of the service provider. It is assumed that a physical connection to the network is present. This step is depicted in Fig. 2 where the **Mobile Station** sends a **DHCP Request** to the **Wireless Router** and receives a **DHCP Answer** containing the IP address. The mobile station is assumed to be equipped with either the domain name or IP address of the provisioning security gateway and the provisioning GAN controller.

Having obtained an IP address via DHCP, the mobile station can now start negotiating the parameters for the secure tunnel with the provisioning security gateway using IKEv2. This is illustrated in the MSC shown in Fig. 3. This is done in three phases. The first phase is the initial IKEv2 exchange, where the two parties agree on the cryptographic algorithms to use, and exchange Diffie-Hellman values in order to establish a shared key for the rest of the message exchanges. The second phase is the exchange of Extensible Authentication Protocol

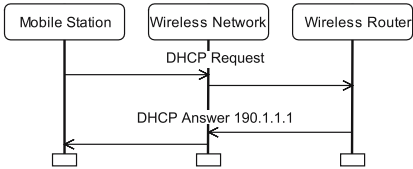


Fig. 2. MSC showing DHCP step of connection establishment

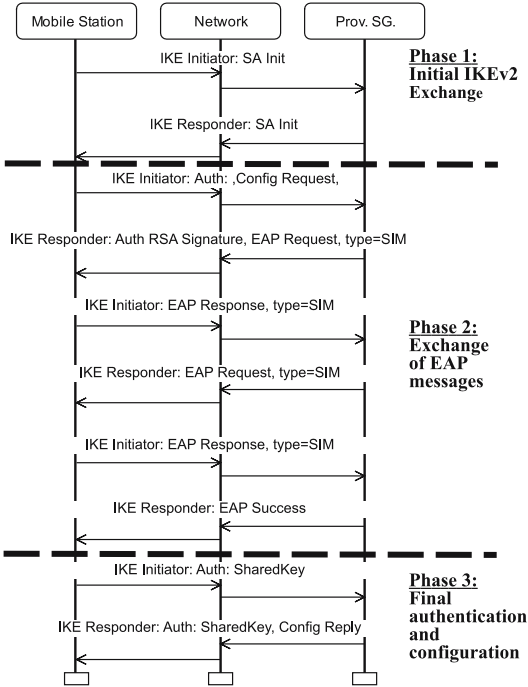


Fig. 3. MSC showing IKE step of connection establishment

(EAP) messages. The idea of EAP is that it is possible to use any kind of authentication protocol with IKEv2. In this situation, a protocol called *EAP-SIM* is used. As can be seen in Fig. 3, the Provisioning Security Gateway initiates the EAP message exchange by returning an *EAP Request* to the Mobile Station’s authentication request. The actual EAP-SIM protocol exchanges 4 messages (2 requests and 2 responses) before it succeeds. As a result of the EAP-phase, the two parties have a shared secret key. In the third phase the Mobile Station uses this shared key to perform final authentication. The last packet sent by the Provisioning Security Gateway contains the network configuration for the Mobile Station needed to establish a secure tunnel.

Having established the secure tunnel, the Mobile Station opens a secure connection to the Provisioning GAN Controller and registers itself. This is shown in

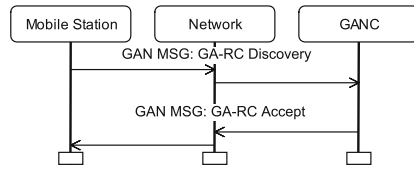


Fig. 4. MSC showing GAN step of connection establishment

the MSC in Fig. 4. If the provisioning GAN controller accepts the mobile station, it sends a redirect message, stating a pair of security gateway and GAN controller to use for any further communication. The **Mobile Station** closes the connection to the **Provisioning GAN Controller** and the **Provisioning Security Gateway**. The final two steps of establishing a connection are to negotiate new IPsec tunnel parameters with the new security gateway, and establish a connection to the GAN controller. Having established the connection, the scenario ends. Fig. 4 only shows the registration with the **Provisioning Security Gateway**.

The scenario modelled involves multiple layers of the IP network stack. DHCP, used to configure the mobile station, is a layer 2 protocol, while IKE is a layer 4 protocol, and GAN is a layer 5 protocol. In order to accommodate all these layers in the model, a rather detailed model of the IP components has been made. However, where simplifications were possible they have been made. For instance, the GAN protocol uses TCP, but TCP has not been modelled. Instead the network is currently loss-less, but may reorder packets. This is not a problem, as GAN messages can be received out of order without problems. This is due to the fact, that the GAN client only receives exactly the message it expects. The IP model contains routing which behaves similarly to the routing procedures implemented in real IP stacks. Some simplifications have been made to the routing algorithm, but the behaviour is the same. Each network node connected to an IP network has a routing table which contains information on how to deliver IP packets. In its simplest form, the entries in a routing table are pairs of destination network address and next hop, describing what the next hop is for IP packets matching the network address.

In the CPN model, IP addresses assigned to local interfaces have entries in the routing table as well, with a special next hop value. This is usually not the case for IP stacks, but simplifies the routing model as ingoing routing can be performed without inspecting the list of interfaces. The ESP tunnel, which is used to secure the communication between the mobile station and the security gateway, is a part of the IPsec protocol suite, which is an extension to the IP stack. Only enough of IPsec is modelled to support this tunnelling, and as such IPsec is not modelled. There are two components that are related to IPsec in the model: Routing and the Security Policy Database (SPD). The routing system ensures that packets are put into the ESP tunnel, and extracted again at the other end. The SPD describes what packets are allowed to be sent and received by the IP stack, and is also responsible for identifying which packets are to be tunnelled. Each entry in the SPD contains the source and destination addresses

to use for matching packets, and an action to perform. Modelled actions are *bypass*, which means allow, and *tunnel*, which means that the matched packet is to be sent through an ESP tunnel.

3 Modelling the GAN Network Nodes

The CPN module is organised in 31 modules, with the top-level module being depicted in Fig. 1. The top module has four submodules: **MobileStation**, **WirelessRouter**, **SecurityGateway** and **GANController**. In this section we present these modules in further detail. It can be seen that the provisioning and default GAN controller is represented using the same module and the same is the case with the provisioning and default security gateways. Each of the modules has one or more protocol modules and an IP layer module. The modelling of the protocol entities and the IP layer will be discussed in Sect. 4.

3.1 Mobile Station

Figure 5 shows the **MobileStation** module. The **IP Layer** substitution transition represents the IP layer of the mobile station and the **Physical Layer** substitution transition represents the interface to the underlying physical network. To the left are the three places which configure the IP layer module with a **Security Policy Database**, a **Routing Table**, and IP and MAC Addresses of the mobile station. These are also accessed in the DHCP, IKE, and GAN modules, as the configuration places are fusion places. This has been done to reduce the number of arcs in the module and since the security policy database, routing table, and addresses are globally accessible in the mobile station. The remaining substitution transitions model the steps that the mobile station goes through when establishing a GAN connection. The mobile station is initially in a **Down** state represented by a unit token on the place **Down**.

There are two tokens on the **Addresses** place representing the MAC and IP addresses assigned to the interfaces of the mobile station. The **ADDR** colour set is defined as follows:

```
colset IP_ADDR      = product INT * INT * INT * INT;
colset MAC_ADDR     = INT;
colset IFACE        = INT;
colset IFACExIP_ADDR = product IFACE * IP_ADDR;
colset IFACExMAC_ADDR = product IFACE * MAC_ADDR;
colset ADDR         = union IpAddr  : IFACExIP_ADDR +
                          MacAddr  : IFACExMAC_ADDR;
```

IP addresses are represented as a product of four integers, one for each octet of the address it represents. So, the IP address 192.0.0.1 becomes (192,0,0,1). MAC addresses and interfaces are represented as integers. As examples, in Fig. 5, **IpAddr**((0,(0,0,0,0))) means that interface 0 is assigned the all-zero IP address ((0,0,0,0)) and **MacAddr**((0,2)) means that interface 0 has the MAC address 2.

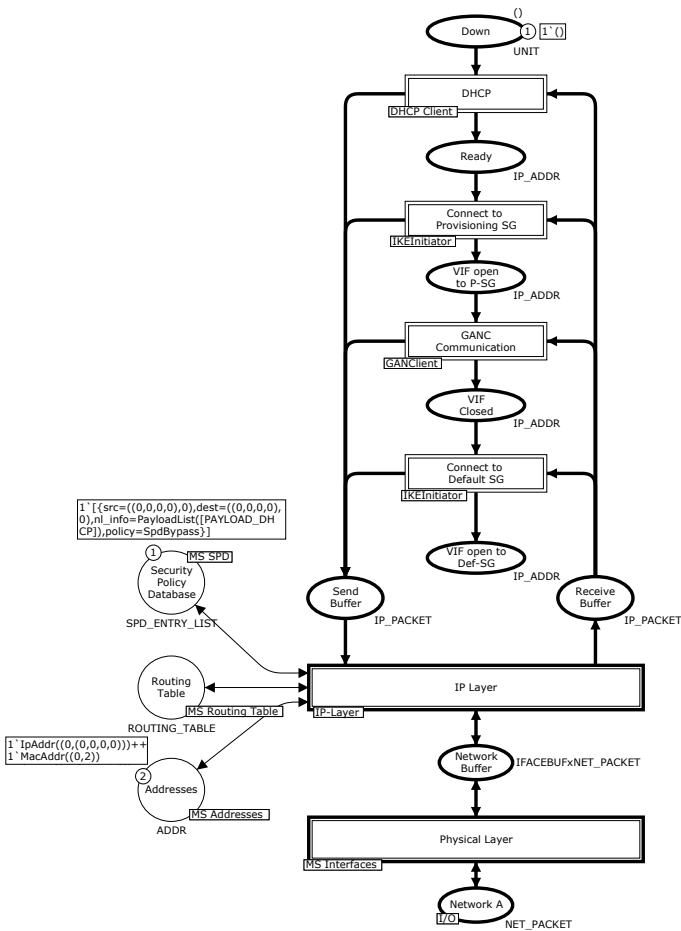


Fig. 5. The MobileStation module

Initially, the SPD is configured to allow DHCP messages to pass in and out of the mobile station. The single token on the Security Policy Database place represents a single rule, matching packets with any source and any destination (`src=((0,0,0,0),0)` and `dest=((0,0,0,0),0)`) and DHCP payload (`nl_info=PayloadList([PAYLOAD_DCHP])`). The policy for this rule is *bypass*, meaning that packets matching this rule are allowed. As the routing table is initially empty (no IP configured), there are no tokens on place **Routing Table**.

The first step is to perform DHCP configuration as was previously illustrated in Fig. 2. This is done in the submodule of the DHCP substitution transition in Fig. 5. The DHCP module accesses all three configuration places. After having configured the mobile station with DHCP, a token is placed on the **Ready** place, representing that the mobile station is now ready to start to communicate with the provisioning security gateway. The **Connect to provisioning SG** substitution

transition takes care of establishing the ESP tunnel to the provisioning security gateway, as shown in the MSC on Fig. 3. After having connected to the provisioning security gateway, the **GAN Communication** transition is responsible for sending a GAN discovery message to the provisioning GAN controller and receiving the answer, which will be either reject or accept. In the first case, the mobile station keeps sending discovery messages until one is accepted. When an accept message is received, the ESP tunnel to the provisioning security gateway is closed, and the IP address of the security gateway in the accept packet is placed on the **VIF Closed** place. Finally, the **Connect to Default SG** transition is responsible for establishing a new ESP tunnel to the default security gateway (which was the one received with the GAN accept message).

3.2 Wireless Router

Figure 6 shows the **WirelessRouter** module. The wireless router has an IP layer, a physical layer, a security policy database, a routing table, and a set of associated addresses similar to the mobile station. The SPD is setup to allow any packets to bypass it. The wireless router has two interfaces, the **Addresses** place assigns MAC address 1 and IP address 190.1.1.254 to interface 0, and MAC address 3 and IP address 172.1.1.254 to interface 1.

The routing table contains a single token of the colour set **ROUTING_TABLE** which represents the complete routing table. This colour set is defined as:

```
colset NETWORK_ADDR = product IP_ADDR * INT;
colset ROUTING_ENTRY = product NETWORK_ADDR * IP_NEXTHOP;
colset ROUTING_TABLE = list ROUTING_ENTRY;
```

The colour set is a list of **ROUTING_ENTRY**. The **NETWORK_ADDR** colour set represents a network address in an IP network. It consists of an IP address

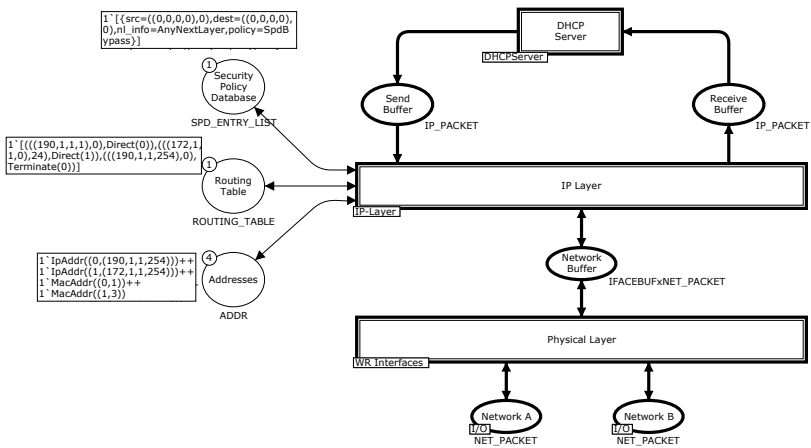


Fig. 6. The WirelessRouter module

and a prefix, which selects how much of the IP address to use for the network address. For instance, a prefix of 24 means that the first 24 bits of the IP address constitute the network address, which corresponds to using the first 3 octets ($3 * 8 = 24$). Usually, this is written as 192.2.0.0/24, but in our model it becomes $((192, 2, 0, 0), 24)$. The IP_NEXTHOP field is explained in Sect. 4.4.

In the Wireless Router module, the routing table is set up such that packets to the host with IP address 190.1.1.1 are to be delivered directly via interface 0 (**Direct(0)**), packets to the network 172.1.1.0/24 are to be delivered directly through interface 1 (**Direct(1)**), and finally packets destined for 190.1.1.254 (the Wireless Router itself) are terminated at interface 0 (**Terminate(0)**).

The wireless router implements the DHCP Server which is used initially by the mobile station to obtain an IP address. It can be seen that the wireless router has a physical connection to both the Wireless Network and Network B.

3.3 Security Gateway

Figure 7 shows the SecurityGateway module. The security gateway has an IP layer, a physical layer, a security policy database, routing table, and a set of associated addresses similar to the mobile station and the wireless router. The configuration places are initialised via the **Init** transition which obtains the configuration parameters from the input port place **Config**. The security gateway implements the IKE Responder protocol module which communicates with the IKE Initiator of the mobile station as described by the MSC shown in Fig. 3.

The **Config** place is associated with the SG Config socket place of the top-level module (see Fig. 1) for the instance of the SecurityGateway module that corre-

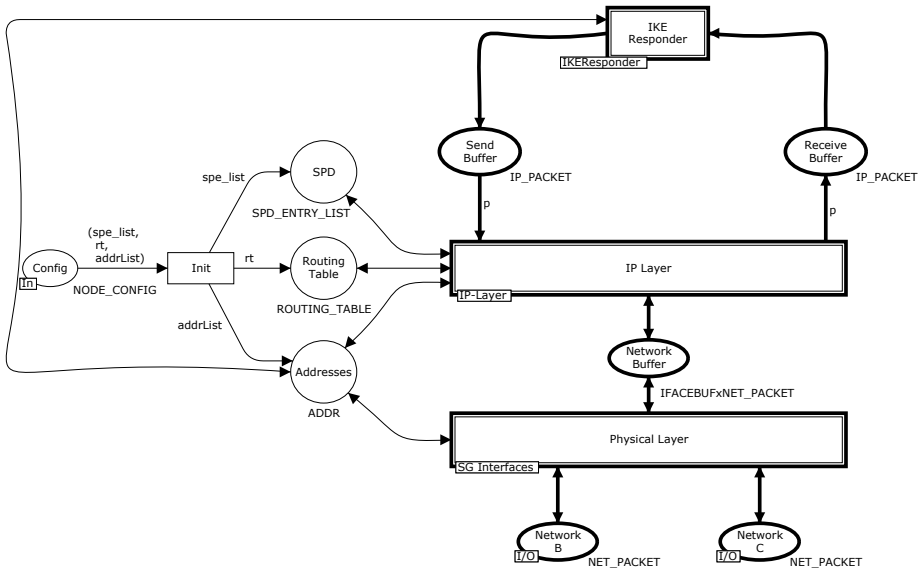


Fig. 7. The SecurityGateway module

sponds to the **Provisioning Security Gateway** substitution transition (see Fig. 1). The initial marking of the **SG Config** configures the provisioning security gateway with two physical interfaces and a virtual interface for use with the ESP-tunnel. Interface 0 is configured with MAC address 4 and IP address 172.1.1.1, while interface 1 is configured with MAC address 5 and IP address 12.0.0.1. The third interface, interface 2, does not have any MAC address assigned to it, but only the IP address 80.0.0.1. The reason for this is that it is a virtual interface. The security policy database is set up such that packets sent via interface 2 are put into an ESP-tunnel. The default security gateway is configured similarly.

3.4 GAN Controller

Figure 8 shows the **GAN Controller** module which is the common submodule of the substitution transitions **Provisioning GAN Controller** and **Default GAN Controller** in Fig. 1. Besides the IP and physical network layers, the **GAN Controller** implements the **GANServer** module to be presented in the next section. The **GAN controllers** are configured in a similar way as the security gateways.

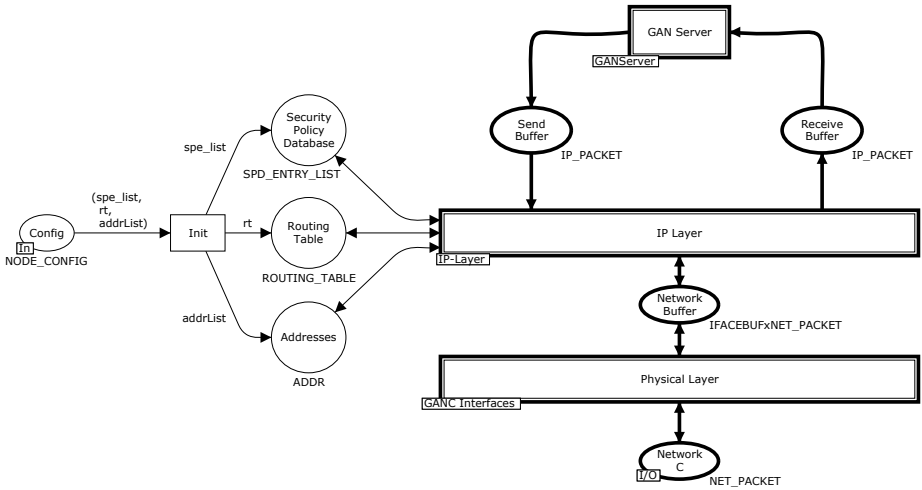


Fig. 8. The GAN Controller module

4 Modelling the Protocol Entities

This section describes the modelling of the protocol entities in the mobile station, wireless router, security gateways, and GAN controllers. The description of the protocol entities has been organised such that it reflects how we have decided to organise the protocol entities in the CPN model. This means that we always describe peer protocol entities, i.e., when describing the DHCP client of the mobile station, we simultaneously describe its peer protocol entity which is the DHCP server of the wireless router.

4.1 Dynamic Host Configuration Protocol

Figure 9 (top) shows the DHCP Client module of the mobile station and Fig. 9 (bottom) shows the DHCP Server module of the wireless router. The two modules model the part of GAN connection establishment which was shown in Fig. 2.

The DHCP client starts by broadcasting a request for an IP address and then awaits a response from the DHCP server. When the DHCP server receives a request it selects one of its **FreeAddresses** and sends an answer back to the DHCP client. When the client receives the answer it configures itself with the IP address and updates its routing table and security policy database accordingly. Three entries are added to the routing table: $1'((\#ip(da), 32), \text{Terminate}(0))$ means that the IP address received in the DHCP answer belongs to interface 0, while $1'(\text{calcNetwork}(\#ip(da), \#netmask(da)), \text{Direct}(0))$ specifies that the network from which the IP address has been assigned, is

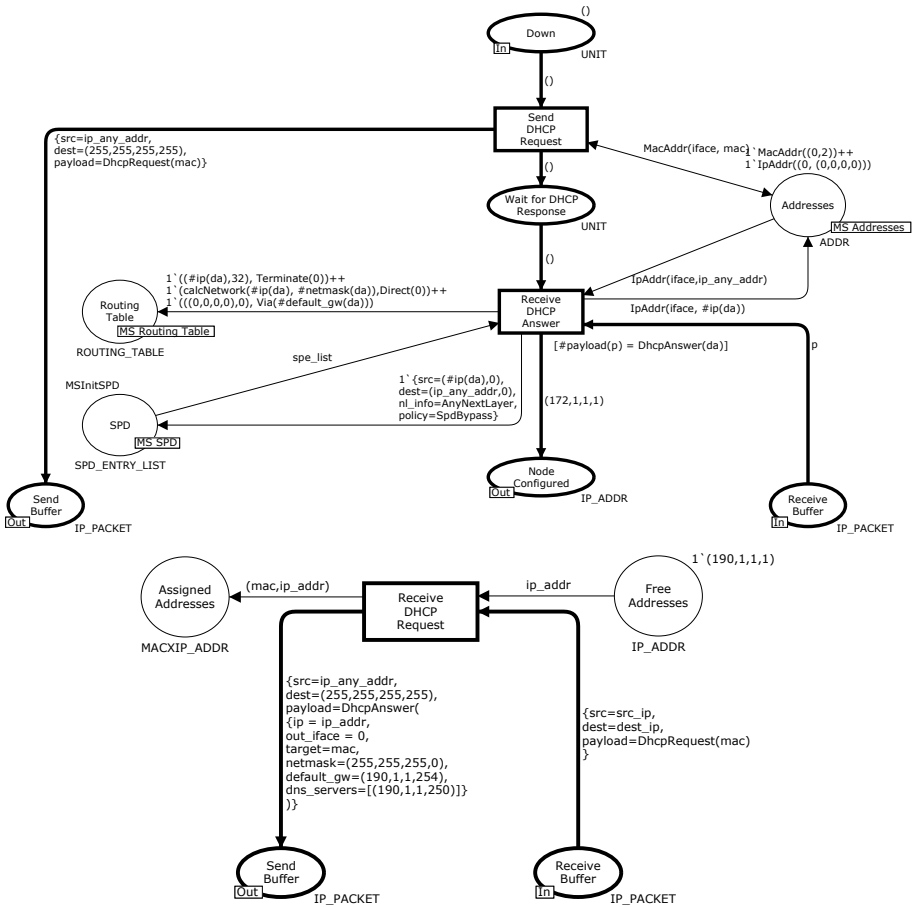


Fig. 9. DHCP client (top) and DHCP server (bottom) modules

reachable via interface 0. Finally, a default route is installed with $1'((0,0,0,0),0)$, $\text{Via}(\#default\_gw(da))$, such that packets which no other routing entry matches, are sent to the default gateway specified in the DHCP answer. The SPD is modified so that the previous rule (which allowed all DHCP traffic) is removed and replaced with a new rule, which states that all packets sent from the assigned IP address are allowed to pass out, and all packets sent to the assigned IP address are allowed to pass in.

4.2 IKEv2 Modules

Figure 10 (left) shows the `IKEInitiator` module of the mobile station and Fig. 10 (right) shows the `IKEResponder` module of the security gateways. The modules model the second step of the GAN connection establishment illustrated in Fig. 3. Each module describes the states that the protocol entities go through

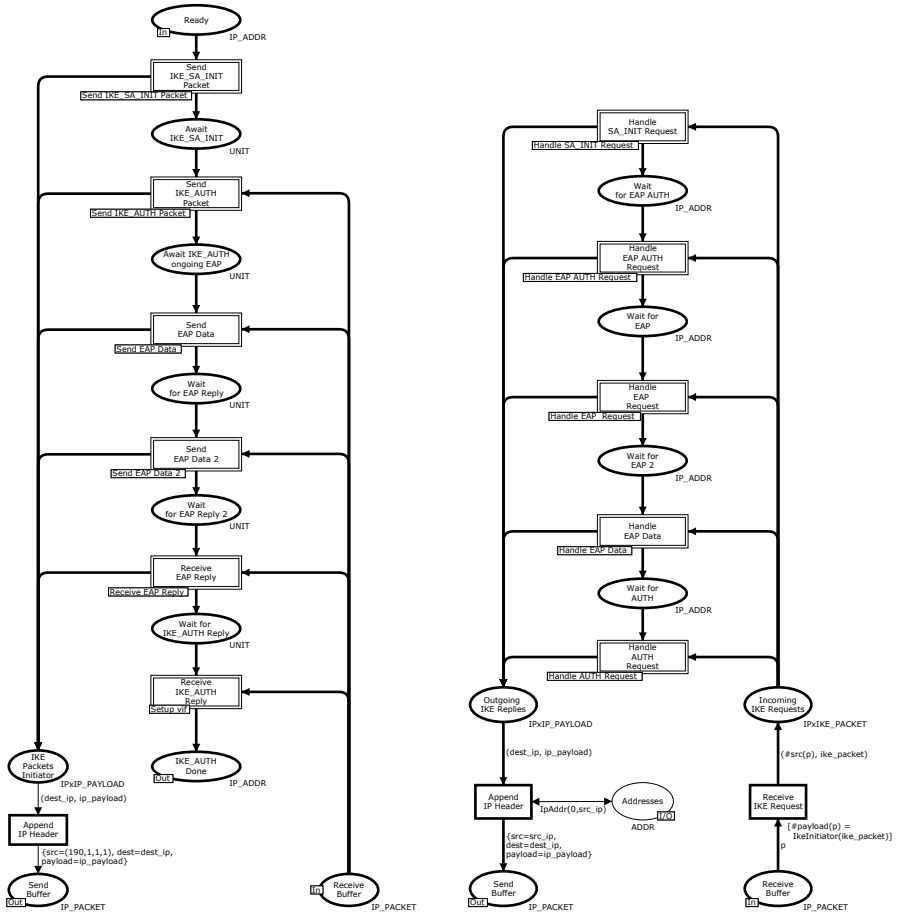


Fig. 10. IKE initiator (left) and IKE responder (right) modules

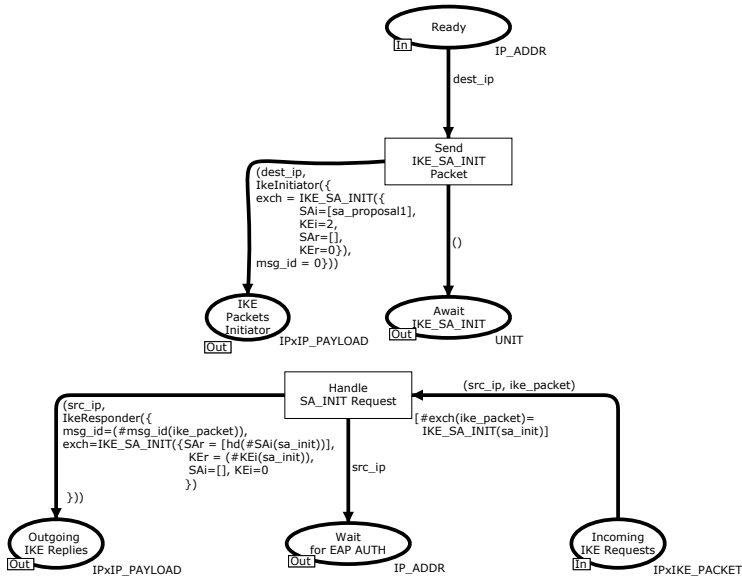


Fig. 11. Example of IKE initiator (top) and IKE responder (bottom) submodules

during the IKE message exchange. The state changes are represented by substitution transitions and Fig. 11 shows the `Send IKE_SA_INIT Packet` and `Handle_SA_INIT_Request` modules.

The `Send IKE_SA_INIT Packet` transition on Fig. 11 (top) takes the IKE Initiator from the state `Ready` to `Await IKE_SA_INIT` and sends an IKE message to the security gateway initialising the communication. The IP address of the security gateway is retrieved from the `Ready` place. Figure 11 (bottom) shows how the `IKE_SA_INIT` packet is handled by the IKE Responder module (which the security gateway implements). Upon receiving an `IKE_SA_INIT` packet it sends a response and moves the responder to the `Wait for EAP Auth` state. The submodules of the other substitution transitions of the IKE modules are similar. Neither initiator nor responder will receive packets that are not expected. They remain in the network buffer forever.

4.3 GAN Modules

Figure 12 (top) shows the `GANClient` module of the mobile station and Fig. 12 (bottom) shows the `GANServer` module of the GAN controller. In the `Tunnel Configured` state, a secure tunnel to the security gateway has been established. The mobile station initiates the GAN communication by sending a GA-RC discovery message. The `Send GA RC Discovery Message` transition does just that, and places the mobile station in the `Wait for GA RC Response` state, where the mobile station will wait until a response is received from the GAN controller. As can be seen in Fig. 12(bottom), the GAN controller can answer a request with

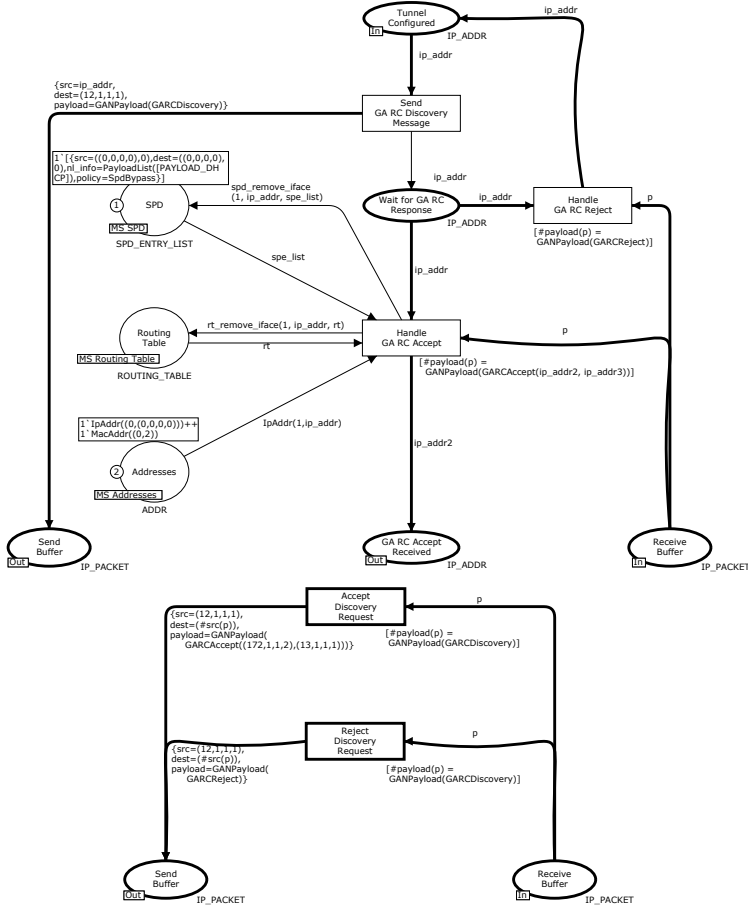


Fig. 12. GAN controller modules mobile station (top) and controller (bottom)

either an accept or reject message. If the GANClient receives a reject response, the Handle GA RC Reject transition will put the client back into the Tunnel Configured state, and another discovery message will be sent. This will continue until an accept message is received, in which case the Handle GA RC Accept transition puts the client in the GA RC Accept Received state, and closes the secure tunnel. This is done by removing the address associated with the tunnel from the Addresses place, and removing any entries in the SPD and routing table containing references to the interface.

4.4 IP Network Layer

Figure 13 shows the IPLayer module which is used to model the IP network layer in the mobile station, wireless router, security gateways, and the GAN controllers. As mentioned in Sect. 2, many details of the IP stack have been

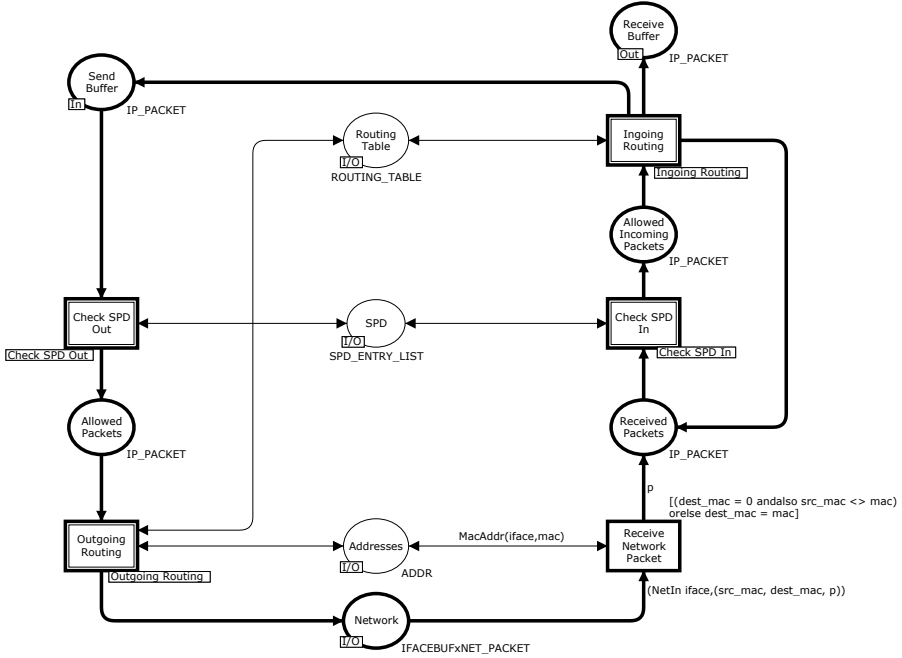


Fig. 13. The IP-Layer module with ingoing and outgoing packet flow

modelled, such as the routing system and the security policy database. Transport protocols (e.g., TCP) are, however, not modelled.

The access to the underlying physical network is modelled via the input/output port place `Network` which has the colour set `NET_PACKET` defined as:

```
colset NET_PACKET = product MAC_ADDR * MAC_ADDR * IP_PACKET;
```

The first component of the product is the source MAC address, while the second is the destination MAC address. The final component is the payload, which currently can only be an IP packet. MAC addresses are represented by integers. The actual physical network layer has been abstracted away, as we do not care whether the physical network is, e.g., an Ethernet or an ATM network.

The chosen representation of a network cannot lose packets but can reorder them, as tokens are picked non-deterministically from the `Network` place. A real network does occasionally drop packets, and considering losing packets is especially important when building fault-tolerant systems. Both IKE and TCP use timeouts and sequence numbers to retransmit packets, such that the protocols deal with lossy networks. Rather than modelling the retransmission schemes, we have chosen to have a loss-less network.

Receiving packets from the network is modelled as a transition which consumes a `NET_PACKET`-token, with the destination MAC address corresponding to the MAC address of the network node. Sending packets is a matter of producing

a `NET_PACKET`-token with the correct source and destination addresses and the intended IP packet. The IP layer is completely generic and configured with three places for network addresses, routing table, and Security Policy Database (*SPD*). The *Routing Table* place corresponds to the routing table found in real IP implementations. It consists of a number of entries, each of a pair (`NETWORK_ADDR`, `IP_NEXTHOP`), where `IP_NEXTHOP` is as defined as:

```
colset IP_NEXTHOP = union Direct    : IFACE    +
                          Via        : IP_ADDR  +
                          Terminate  : IFACE;
```

The colour set defines which action is to be taken for packets going to the specified IP network. There are three possible actions: **Direct** (the packet can be delivered directly via the local physical network via `IFACE`), **Via** (the packet can get closer to its destination by being delivered to the `IP_ADDR`), and **Terminate** (the destination network address is a local interface address of `IFACE`).

The *SPD* place is the *SPD* database, and describes which packets are allowed to be sent and received by the IP layer. An entry in the *SPD* database can also specify that certain packets are to be tunnelled through a secure tunnel. Finally, the *Addresses* place contains a token for each address known to the IP Layer. These addresses are both physical (MAC) addresses and IP addresses. Each `ADDR` token contains an interface number and a MAC or IP address.

Packets to be sent are put on the *Send Buffer* place by upper-layers. The first step done by the IP layer, is to check the *SPD* database, which is done by the *Check SPD Out* transition. The *Check SPD Out* module inspects the *SPD* database to find an entry which matches the source and destination addresses of the packet. If no entry is found, the packet is dropped. If an entry is found, the action of the entry is applied. Either the action is *bypass*, meaning that the packet can be sent further down the IP-stack, or the action is *tunnel*, meaning that the packet is to be sent through a secure tunnel. In the latter case, a new IP-packet is constructed according to the tunnel information associated with the *tunnel* action. If the packet is allowed to be sent, it is put on the *Allowed Packets* place. In order to be able to construct a `NET_PACKET` token, the destination MAC needs to be found. This is done by the *Outgoing Routing* transition. As can be seen in Fig. 13, it needs to know both about the routing table and the addresses. If a next-hop has been found, a token is placed on the *Network* place with information about which interface to send on and the MAC address of the next-hop.

Ingoing packets are retrieved from the *Network* place by the *Receive Network Packet* transition. Destination MAC and interface number of the network packet have to match one of the MAC addresses configured in the *Addresses* place. The `IP_PACKET` inside the `NET_PACKET` is placed on the *Received Packets* place. *Check SPD In* performs incoming *SPD* check, while *Ingoing Routing* decides if the packet is to be delivered locally, forwarded, or is a tunnelled packet which has to be de-tunnelled. In the latter case, the packet is put on the *Received Packets* place, and goes through *SPD* checking and routing again. If the packet has its final

destination at the local node, it is put in the Receive Buffer. The `IP_PACKET` colour set models IP packets and is defined as:

```
colset IP_PACKET = record dest      : IP_ADDR *
                        src          : IP_ADDR *
                        payload      : IP_PAYLOAD;
```

It has a source and destination IP address and a payload. The `IP_PAYLOAD` colour set is a union of all possible payloads. The colour set is never used by any of the generic network components, and is as such defined accordingly to the rest of the model. In the GAN scenario, the `IP_PAYLOAD` colour set has constructors for DHCP, IKEv2, and GAN packets.

5 Validation of the GAN Scenario

During the construction of the model, simulation was used to check that the model behaviour was as desired. Even though simulation does not guarantee correct behaviour, it was very useful in finding modelling errors. For instance, simulation was a valuable tool in validating the packet forwarding capabilities of the IP layer modules. By placing tokens that represent packets to be forwarded on a network node's input buffer and starting the simulation, it is easy to see if the packet is being placed in the output buffer as expected. If not, then single-stepping through the simulation helps to understand the behaviour of the model, and modify it so that it behaves as intended. Simulation was also effective in making explicit where further specification of the message exchanges were required, i.e., where the GAN specification was not sufficiently detailed. Furthermore, simulation was heavily used to show engineers at TietoEnator, how the model and especially how the IP-packet flow worked and thereby enabling discussions of the specification. The advantage of simulation over state space verification is that the simulation has immediate visual feedback, and as such is much easier to understand.

A formal validation was performed on the state space generated from the model. The generated state space consists of 3,854 nodes, while the Strongly Connected Component (SCC) graph consists of 3,514 nodes of which 1 is non-trivial, i.e., consists of more than a single state space node. The state space has a single home marking which is also a dead marking. Hence, this marking can always be reached and it has no enabled transitions.

The most interesting property to check is that the mobile station can always end up being configured properly, i.e., that it has both gotten an IP address, has successfully communicated with the provisioning GAN controller, and received addresses of the default security gateway and GAN controller. For this, we checked that in the single home and dead marking identified, there is a token in the `VIF open to Def-SG` place of the mobile station (see Fig. 5). Furthermore, we checked that there were no tokens in any of the other places of the mobile station state machine. This would indicate an error in the model, as we do not want the mobile station to be able to be in two states at the same time. To do this we

defined a predicate, which checks that only the VIF open to Def-SG contains tokens. Checking all nodes in the state space for this predicate, shows that it holds only for the single home and dead marking. It is also interesting to investigate whether the routing table and security policy database look as expected. Rather than defining predicates, we displayed the dead marking in the simulator tool of CPN Tools and inspected the configuration of the mobile station. This showed that both routing tables, address assignments, and security policy database were as expected. The state space report generated by CPN Tools also showed that the transitions `RejectDiscoveryRequest` and `HandleGARCReject` (see Fig. 12) both were impartial. This means that if the system does not terminate in the home and dead marking discussed above, then it is because the GAN controller keeps rejecting the connection.

6 Conclusion and Future Work

The overall goal of the project was to construct a CPN model and obtain a more complete specification of the GAN scenario to be implemented by TietoEnator. The CPN model presented in this paper matches the GAN scenario closely, meaning that every entity in the scenario has been represented in the model, and every action in the scenario has a model counterpart. The act of constructing the CPN model helped to specify the details of the message exchanges that were not explicit in the textual scenario description. Including a detailed modelling of the IP stack in the CPN model was necessary in order to capture the details of sending GAN packets from the mobile station to the GAN controller. Furthermore, it was required in order to validate correctness of the the routing table contents, SPD entries, and IP address distribution. The CPN model was discussed with TietoEnator engineers, and jointly validated using the simulation capabilities of CPN Tools. Further application of simulation and state space analysis has helped to obtain further confidence in the constructed model, and more importantly it has provided valuable information about the properties of the scenario.

In near future, TietoEnator is going to implement the GAN controller. Based on the experience from the project presented in this paper, it has been decided that CPNs will be used to model the initial controller design. Besides the advantages of having a formal model which can be validated by means of state space analysis, the goal is to generate template code for the GAN controller directly from a CPN model of the controller. This will ease the work of the initial implementation and help ensure that the implementation is consistent with the design as specified by the CPN model. Initial work in this direction is presented in [10].

References

1. 3GPP. Digital cellular telecommunications system (Phase 2+); Generic access to the A/Gb interface; Stage 2. 3GPP TS 43.318 version 6.9.0 Release 6 (March 2007)
2. 3GPP. Website of 3GPP (May 2007), <http://www.3gpp.org>

3. Billington, J., Han, B.: Modelling and Analysing the Functional Behaviour of TCP Connection Management Procedures. *International Journal on Software Tools for Technology Transfer* 9(3-4), 269–304 (2007)
4. CPN Tools, <http://www.daimi.au.dk/CPNTools>
5. TietoEnator Denmark, <http://www.tietoenator.dk>
6. Droms, R.: Dynamic Host Configuration Protocol. RFC2131 (March 1997)
7. Kaufman, C. (ed.): Internet Key Exchange Protocol. RFC4306 (December 2005)
8. Fleischer, P.: Towards a Formal Specification of a Generic Access Network Architecture using Coloured Petri Nets. In: *Proc. of Workshop on Petri Nets and Software Engineering (PNSE 2007)*, pp. 231–232 (2007)
9. Fleischer, P., Kristensen, L.M.: Towards Formal Specification and Validation of Secure Connection Establishment in a Generic Access Network Scenario. In: Fleischer, P., Kristensen, L.M. (eds.) *Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, October 2007. DAIMI PB, vol. 584, pp. 9–28 (2007)
10. Fleischer, P., Kristensen, L.M.: Modelling of the Configuration/Management API Middleware using Coloured Petri Nets. In: *Proc. of PNTAP 2008* (2008)
11. Grimstrup, P.: Interworking Description for IKEv2 Library. Ericsson Internal. Document No. 155 10-FCP 101 4328 Uen (September 2006)
12. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer* 9(3-4), 213–254 (2007)
13. Kent, S.: IP Encapsulating Security Payload (ESP). RFC4303 (December 2005)
14. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC4301 (December 2005)
15. Kristensen, L.M., Jensen, K.: Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networks. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) *INT 2004*. LNCS, vol. 3147, pp. 248–269. Springer, Heidelberg (2004)
16. Vanit-Anunchai, S., Billington, J.: Modelling the Datagram Congestion Control Protocol's Connection Management and Synchronization Procedures. In: Kleijn, J., Yakovlev, A. (eds.) *ICATPN 2007*. LNCS, vol. 4546, pp. 423–444. Springer, Heidelberg (2007)
17. Westergaard, M., Lassen, K.B.: The BRITNeY Suite Animation Tool. In: Donatelli, S., Thiagarajan, P.S. (eds.) *Petri Nets and Other Models of Concurrency - ICATPN 2006*. LNCS, vol. 4024, pp. 431–440. Springer, Heidelberg (2006)

Parametric Language Analysis of the Class of Stop-and-Wait Protocols

Guy Edward Gallasch and Jonathan Billington

Computer Systems Engineering Centre
University of South Australia
Mawson Lakes Campus, SA, 5095, Australia
{guy.gallasch,jonathan.billington}@unisa.edu.au

Abstract. Model checking a parametric system when one or more of its parameters is unbounded requires considering an infinite family of models. The Stop-and-Wait Protocol (SWP) has two (unbounded) parameters: the maximum sequence number and the maximum number of retransmissions. Previously, we presented a novel method for the *parametric analysis* of the SWP by developing algebraic formulas in the two parameters that symbolically represent the corresponding infinite class of reachability graphs. Properties were then verified directly from these expressions. This paper extends this analysis to the verification of the SWP using language equivalence. From the algebraic expressions developed previously, a parametric Finite State Automaton (FSA) representing all sequences of user-observable events (i.e. the protocol language) is derived. We then perform determinisation and minimisation directly on the parametric FSA. The result is a simple, non-parametric FSA that is isomorphic to the service language of alternating send and receive events. This result is significant as it verifies conformance of the SWP to its service for all values of the two unbounded parameters.

1 Introduction

In [6] we presented a parametric analysis of the class of Stop-and-Wait protocols in two parameters: the Maximum Sequence Number (**MaxSeqNo**), and the Maximum Number of Retransmissions (**MaxRetrans**). From a Coloured Petri Net [10, 11] model of the Stop-and-Wait Protocol, parameterised by **MaxSeqNo** and **MaxRetrans**, algebraic expressions representing the infinite class of reachability graphs were developed and proved correct. From these algebraic expressions a number of properties were verified directly, including the absence of deadlock, absence of livelock, and absence of dead transitions. The channel bounds and size of the parametric reachability graph in terms of the two parameters were also determined.

This paper continues the work in [6] by considering the next step in protocol verification [2], that of *language analysis*. A protocol will exhibit certain behaviour to the users of that protocol and this should conform to the expected behaviour of the protocol. This expected behaviour is called the *service specification*, which includes the *service language*, the set of all allowable sequences of

service primitives (user-observable actions). The sequences of service primitives exhibited by the protocol is known as the *protocol language*. If the protocol exhibits behaviour that is not in the service language, then this indicates an error in the protocol (assuming the service is correct).

In [2] the concrete reachability graph for a particular instantiation of the protocol is mapped to a Finite State Automaton (FSA) that represents the protocol language. In this paper we generalise this step by mapping the parametric reachability graph to a parametric FSA. We then apply FSA reduction techniques [1, 9, 12] through direct manipulation of the algebraic expressions of the parametric FSA. We find that the epsilon-removal and determinisation procedure removes **MaxRetrans**, and the minimisation procedure removes **MaxSeqNo**. The result is a single, simple non-parametric FSA of the protocol language, which conforms to the service language. Previously [5] we had only been able to obtain this result for one parameter, the case when **MaxRetrans**=0.

The paper provides two main contributions. Firstly, we extend the parametric verification approach to parametric language analysis in both parameters. Secondly, we verify that the Stop-and-Wait Protocol conforms to its service of alternating send and receive events for *all* values of the two unbounded parameters. The authors are not aware of any previous attempts to obtain an explicit algebraic representation for the infinite family of FSAs representing the protocol language of the Stop-and-Wait Protocol in both parameters, or to perform automata reduction on such an algebraic representation.

2 Parametric Reachability Graph

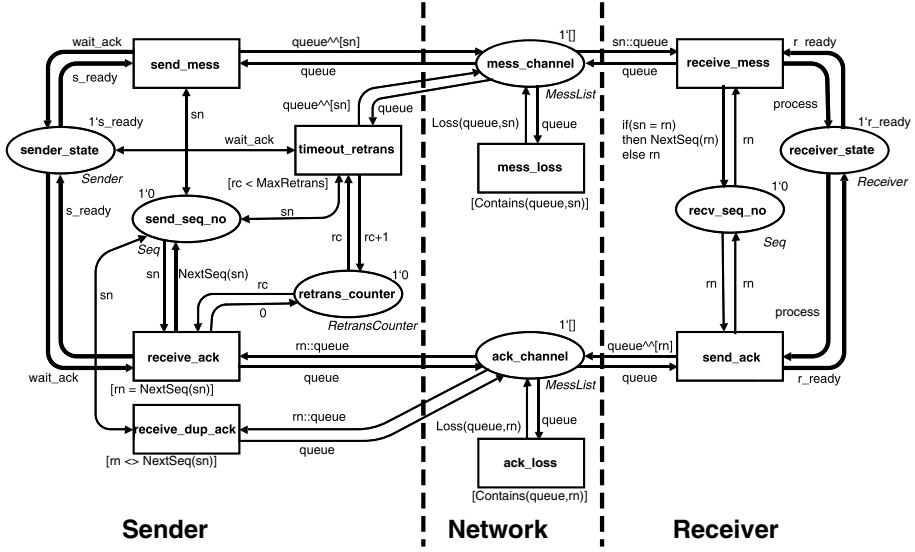
In [5, 6] we presented a parameterised Coloured Petri Net (CPN) model of the Stop-and-Wait Protocol (SWP) and in [6] derived notation for the markings and arcs of its reachability graph (RG). We then stated and proved correct a set of algebraic expressions representing the parametric reachability graph (PRG) in both parameters.

In this section we recall our parametric SWP CPN model, notation for markings and arcs, and then the parametric reachability graph itself. In this paper we do not describe the SWP CPN model in detail, nor the derivation of the marking and arc notation. Full details can be found in [4, 7, 8].

2.1 Stop-and-Wait Protocol CPN Model

The SWP CPN model, shown in Fig. 1, consists of 7 places and 8 transitions and is self explanatory to those familiar with CPNs. The model in its current form was first presented and described in [5], with the exception of the model of loss, which was generalised in [6] to allow loss of messages and acknowledgements from anywhere in the in-order channels.

The initial marking is given above each place, to the left or right as room permits. The notation $1'x$ is used to represent the multiset containing one instance of the single element, x . Intuitively, both the sender and receiver start



```

val MaxRetrans = 0;
val MaxSeqNo = 1;

color Sender = with s_ready | wait_ack;
color Receiver = with r_ready | process;
color Seq = int with 0..MaxSeqNo;
color RetransCounter = int with 0..MaxRetrans;
color Message = Seq;
color MessList = list Message;

var sn, rn : Seq;
var rc : RetransCounter;
var queue : MessList;

fun NextSeq(n) =
  if (n = MaxSeqNo)
  then 0 else n+1;

fun Contains([], sn) = false
| Contains(m::queue, sn) =
  if (sn=m)
  then true else Contains(queue, sn);

fun Loss(m::queue, sn) =
  if (sn=m) then queue
  else m::Loss(queue, sn);

```

Fig. 1. The CPN Diagram and Declarations of our SWP CPN Model

in their respective ready states, with sequence numbers of 0. The message and acknowledgement channels are both empty, and the retransmission counter at the sender is initially 0.

2.2 Notational Conventions

In CPNs, a transition coupled with a binding of its variables to values is known as a *binding element* or *transition mode*. We use the notation $[M]$ to represent the set of markings (states) reachable from M (note that $M \in [M]$). The notation $M[(t, b)]$ indicates that the binding element, (t, b) , is enabled in M , where t is the name of the transition and b is a binding of the variables of the transition to values. In addition, we use the following notation:

- $MS \in \mathbb{N}^+$ and $MR \in \mathbb{N}$ are used as shorthand for MaxSeqNo and MaxRetrans respectively;
- i^j is used to represent j repetitions of sequence number i ;

- $\oplus_{MS}$ is used to represent modulo $MS + 1$ addition; and
- $\ominus_{MS}$ is used to represent modulo $MS + 1$ subtraction.

Definition 1 (Reachability Graph). *The RG of a CPN with initial marking M_0 and a set of binding elements BE , is a labelled directed graph $RG = (V, A)$ where*

1. $V = [M_0]$ is the set of reachable markings of the CPN; and
2. $A = \{(M, (t, b), M') \in V \times BE \times V \mid M[(t, b)]M'\}$ is the set of labelled directed arcs, where $M[(t, b)]M'$ denotes that the marking of the CPN changes from M to M' on the occurrence of binding element $(t, b) \in BE$, where t is the name of a transition and b is a binding of the variables of the transition.

The parameterised SWP CPN and its RG are denoted by $CPN_{(MS, MR)}$ and $RG_{(MS, MR)} = (V_{(MS, MR)}, A_{(MS, MR)})$ respectively.

2.3 Marking and Arc Notation

A notation for markings, derived in [6, 7], is as follows. For $CPN_{(MS, MR)}$ a marking $M \in V_{(MS, MR)}$ is denoted by $M_{(class, i), (mo, ao, mn, an, ret)}^{(MS, MR)}$ where the superscript contains the parameter values of the SWP CPN and the subscript contains the marking description, where:

- $(class, i)$ encodes the marking of the `sender_state` and `receiver_state` places (sender and receiver ‘major’ state), and the receiver sequence number for sender sequence number, i , as defined in Table 1;
- $mo, mn \in \mathbb{N}$ encode the number of instances of the previously acknowledged (old) message (with sequence number $i \ominus_{MS} 1$) and the currently outstanding (new) message (with sequence number i), respectively, in the message channel;
- $ao, an \in \mathbb{N}$ encode the number of instances of the acknowledgement of the previous (old) message and of the currently outstanding (new) message, respectively, in the acknowledgement channel; and
- $ret \in RetransCounter$ is the number of times the currently outstanding (unacknowledged) message has been retransmitted.

Thus for a given $M \in V_{(MS, MR)}$ represented by $M_{(class, i), (mo, ao, mn, an, ret)}^{(MS, MR)}$ the marking of places `sender_state`, `receiver_state`, `send_seq_no` and `rcv_seq_no` is defined by Table 1 and:

$$\begin{aligned}
 M(\text{mess\_channel}) &= 1'[(i \ominus_{MS} 1)^{mo} i^{mn}] \\
 M(\text{ack\_channel}) &= 1'[i^{ao} (i \oplus_{MS} 1)^{an}] \\
 M(\text{retrans\_counter}) &= 1'ret
 \end{aligned}$$

Similar notation is defined for arcs in [8] but it suffices for this paper to identify arcs by their source marking and enabled binding element. The destination marking can be obtained by applying the transition rule for CPNs [10].

Table 1. Classification of markings into Classes

$M(\text{sender\_state})$	$M(\text{receiver\_state})$	$M(\text{send\_seq\_no})$	$M(\text{recv\_seq\_no})$	$Class_{MS}(M)$
1's_ready	1'r_ready	1'i	1'i	1
1'wait_ack	1'r_ready	1'i	1'i	2a
1'wait_ack	1'r_ready	1'i	$1'(i \oplus_{MS} 1)$	2b
1'wait_ack	1'process	1'i	1'i	3a
1'wait_ack	1'process	1'i	$1'(i \oplus_{MS} 1)$	3b
1's_ready	1'process	1'i	1'i	4

Sets of markings and sets of arcs are defined as follows:

Definition 2 (Sets of Markings). *Sets of markings are given by:*

1. $V_{(class,i)}^{(MS,MR)} = \{M \in V_{(MS,MR)} \mid Class_{MS}(M) = class, M(\text{send\_seq\_no}) = 1'i\}$
2. $V_i^{(MS,MR)} = \bigcup_{class \in \{1,2a,2b,3a,3b,4\}} V_{(class,i)}^{(MS,MR)}$ represents the set of markings in which the sender sequence number is given by i .

Definition 3 (Sets of Arcs). *Sets of arcs are given by:*

1. $A_{(class,i)}^{(MS,MR)} = \{(M, (t, b), M') \in A_{(MS,MR)} \mid Class_{MS}(M) = class, M(\text{send\_seq\_no}) = 1'i\}$
2. $A_i^{(MS,MR)} = \bigcup_{class \in \{1,2a,2b,3a,3b,4\}} A_{(class,i)}^{(MS,MR)}$

2.4 Algebraic Expressions for the SWP CPN PRG

Algebraic expressions for the sets of markings and arcs of $RG_{(MS,MR)}$ are defined using the notation above by specifying allowable ranges of the five variables, mo , ao , mn , an and ret , in terms of the parameters. All variables are assumed to only take values that are greater than or equal to 0, unless otherwise indicated. The parametric reachability graph over both parameters is given in the following theorem (from [4, 6]):

Theorem 1 (Parametric Reachability Graph of the SWP CPN).

For $MS \in \mathbb{N}^+$, $MR \in \mathbb{N}$, $RG_{(MS,MR)} = (V_{(MS,MR)}, A_{(MS,MR)})$

$$\text{where } V_{(MS,MR)} = \bigcup_{0 \leq i \leq MS} V_i^{(MS,MR)} \text{ and } A_{(MS,MR)} = \bigcup_{0 \leq i \leq MS} A_i^{(MS,MR)}$$

where all nodes and arcs are defined in Tables 2 to 8.

All of the markings of $RG_{(MS,MR)}$ are described in Table 2 by evaluating the expressions in this table for $0 \leq i \leq MS$. Each row defines a new set of markings. The first column gives the name of the set of markings according to its class. Column 2 defines the set of markings by specifying the allowed ranges of variable values.

Table 2. $V_i^{(MS,MR)}$ for $0 \leq i \leq MS$ and $Class = \{1, 2a, 2b, 3a, 3b, 4\}$

Name	Set Definition
$V_{(1,i)}^{(MS,MR)}$	$\{M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR\}$
$V_{(2a,i)}^{(MS,MR)}$	$\{M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR, 0 \leq ret \leq MR, 0 \leq mn \leq ret + 1\}$
$V_{(2b,i)}^{(MS,MR)}$	$\{M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)} \mid 0 \leq ao \leq MR, 0 \leq ret \leq MR, 0 \leq mn \leq ret, 0 \leq mn + an \leq ret + 1\}$
$V_{(3a,i)}^{(MS,MR)}$	$\{\}, \text{ for } MR = 0; \text{ or } \{M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR - 1, 0 \leq ret \leq MR, 0 \leq mn \leq ret + 1\}, \text{ for } MR > 0.$
$V_{(3b,i)}^{(MS,MR)}$	$\{M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)} \mid 0 \leq ao \leq MR, 0 \leq ret \leq MR, 0 \leq mn + an \leq ret\}$
$V_{(4,i)}^{(MS,MR)}$	$\{\}, \text{ for } MR = 0; \text{ or } \{M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR - 1\}, \text{ for } MR > 0.$

Table 3. The set of arcs $A_{(1,i)}^{(MS,MR)}$ with source markings in $V_{(1,i)}^{(MS,MR)}$

Condition	Source Marking	Binding Element	Destination Marking
none	$M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{send\_mess} \langle \text{sn}=i, \text{queue} = [(i \ominus_{MS} 1)^{mo}] \rangle$	$M_{(2a,i),(mo,ao,1,0,0)}^{(MS,MR)}$
$mo \geq 1$	$M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{mess\_loss} \langle \text{sn}=i \ominus_{MS} 1, \text{queue} = [(i \ominus_{MS} 1)^{mo}] \rangle$	$M_{(1,i),(mo-1,ao,0,0,0)}^{(MS,MR)}$
$mo \geq 1$	$M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{receive\_mess} \langle \text{sn}=i \ominus_{MS} 1, \text{rn}=i, \text{queue} = [(i \ominus_{MS} 1)^{mo-1}] \rangle$	$M_{(4,i),(mo-1,ao,0,0,0)}^{(MS,MR)}$
$ao \geq 1$	$M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{ack\_loss} \langle \text{rn}=i, \text{queue} = [i^{ao}] \rangle$	$M_{(1,i),(mo,ao-1,0,0,0)}^{(MS,MR)}$
$ao \geq 1$	$M_{(1,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{receive\_dup\_ack} \langle \text{sn}=i, \text{rn}=i, \text{queue} = [i^{ao-1}] \rangle$	$M_{(1,i),(mo,ao-1,0,0,0)}^{(MS,MR)}$

Table 4. The set of arcs $A_{(2a,i)}^{(MS,MR)}$ with source markings in $V_{(2a,i)}^{(MS,MR)}$

Condition	Source Marking	Binding Element	Destination Marking
$ret < MR$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{timeout\_retrans} \langle \text{sn}=i, \text{rc}=ret, \text{queue} = [(i \ominus_{MS} 1)^{mo} i^{mn}] \rangle$	$M_{(2a,i),(mo,ao,mn+1,0,ret+1)}^{(MS,MR)}$
$mo \geq 1$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{mess\_loss} \langle \text{sn}=i \ominus_{MS} 1, \text{queue} = [(i \ominus_{MS} 1)^{mo} i^{mn}] \rangle$	$M_{(2a,i),(mo-1,ao,mn,0,ret)}^{(MS,MR)}$
$mn \geq 1$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{mess\_loss} \langle \text{sn}=i, \text{queue} = [(i \ominus_{MS} 1)^{mo} i^{mn}] \rangle$	$M_{(2a,i),(mo,ao,mn-1,0,ret)}^{(MS,MR)}$
$mo \geq 1$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{receive\_mess} \langle \text{sn}=i \ominus_{MS} 1, \text{rn}=i, \text{queue} = [(i \ominus_{MS} 1)^{mo-1} i^{mn}] \rangle$	$M_{(3a,i),(mo-1,ao,mn,0,ret)}^{(MS,MR)}$
$mn \geq 1$	$M_{(2a,i),(0,ao,mn,0,ret)}^{(MS,MR)}$	$\text{receive\_mess} \langle \text{sn}=i, \text{rn}=i, \text{queue} = [i^{mn-1}] \rangle$	$M_{(3b,i),(0,ao,mn-1,0,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{ack\_loss} \langle \text{rn}=i, \text{queue} = [i^{ao}] \rangle$	$M_{(2a,i),(mo,ao-1,mn,0,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$\text{receive\_dup\_ack} \langle \text{sn}=i, \text{rn}=i, \text{queue} = [i^{ao-1}] \rangle$	$M_{(2a,i),(mo,ao-1,mn,0,ret)}^{(MS,MR)}$

All of the arcs of $RG_{(MS,MR)}$ are described in a similar way in Tables 3 to 8 by evaluating each table for $0 \leq i \leq MS$. There is one table of arcs for every row of Table 2, describing the set of arcs with a source marking from that set. Thus

Table 5. The set of arcs $A_{(2b,i)}^{(MS,MR)}$ with source markings in $V_{(2b,i)}^{(MS,MR)}$

Condition	Source Marking	Binding Element	Destination Marking
$ret < MR$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$timeout\_retrans < sn=i, rc=ret,$ $queue = [i^{mn}]>$	$M_{(2b,i),(0,ao,mn+1,an,ret+1)}^{(MS,MR)}$
$mn \geq 1$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$mess\_loss < sn=i, queue = [i^{mn}]>$	$M_{(2b,i),(0,ao,mn-1,an,ret)}^{(MS,MR)}$
$mn \geq 1$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$receive\_mess < sn=i, rn=i \oplus_{MS} 1,$ $queue = [i^{mn-1}]>$	$M_{(3b,i),(0,ao,mn-1,an,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$ack\_loss < rn=i,$ $queue = [i^{ao} (i \oplus_{MS} 1)^{an}]>$	$M_{(2b,i),(0,ao-1,mn,an,ret)}^{(MS,MR)}$
$an \geq 1$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$ack\_loss < rn=i \oplus_{MS} 1,$ $queue = [i^{ao} (i \oplus_{MS} 1)^{an}]>$	$M_{(2b,i),(0,ao,mn,an-1,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(2b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$receive\_dup\_ack < sn=i, rn=i,$ $queue = [i^{ao-1} (i \oplus_{MS} 1)^{an}]>$	$M_{(2b,i),(0,ao-1,mn,an,ret)}^{(MS,MR)}$
$an \geq 1$	$M_{(2b,i),(0,0,mn,an,ret)}^{(MS,MR)}$	$receive\_ack < sn=i, rn=i \oplus_{MS} 1,$ $rc = ret, queue =$ $[(i \oplus_{MS} 1)^{an-1}]>$	$M_{(1,i \oplus_{MS} 1),(mn,an-1,0,0,0)}^{(MS,MR)}$

Table 6. The set of arcs $A_{(3a,i)}^{(MS,MR)}$ with source markings in $V_{(3a,i)}^{(MS,MR)}$, for $MR > 0$

Condition	Source Marking	Binding Element	Destination Marking
$ret < MR$	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$timeout\_retrans < sn=i, rc=ret,$ $queue = [(i \ominus_{MS} 1)^{mo} i^{mn}]>$	$M_{(3a,i),(mo,ao,mn+1,0,ret+1)}^{(MS,MR)}$
$mo \geq 1$	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$mess\_loss < sn=i \ominus_{MS} 1,$ $queue = [(i \ominus_{MS} 1)^{mo} i^{mn}]>$	$M_{(3a,i),(mo-1,ao,mn,0,ret)}^{(MS,MR)}$
$mn \geq 1$	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$mess\_loss < sn=i,$ $queue = [(i \ominus_{MS} 1)^{mo} i^{mn}]>$	$M_{(3a,i),(mo,ao,mn-1,0,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$ack\_loss < rn=i, queue = [i^{ao}]>$	$M_{(3a,i),(mo,ao-1,mn,0,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$receive\_dup\_ack < sn=i, rn=i,$ $queue = [i^{ao-1}]>$	$M_{(3a,i),(mo,ao-1,mn,0,ret)}^{(MS,MR)}$
none	$M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$	$send\_ack < rn=i, queue = [i^{ao}]>$	$M_{(2a,i),(mo,ao+1,mn,0,ret)}^{(MS,MR)}$

Table 7. The set of arcs $A_{(3b,i)}^{(MS,MR)}$ with source markings in $V_{(3b,i)}^{(MS,MR)}$

Condition	Source Marking	Binding Element	Destination Marking
$ret < MR$	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$timeout\_retrans < sn=i, rc=ret,$ $queue = [i^{mn}]>$	$M_{(3b,i),(0,ao,mn+1,an,ret+1)}^{(MS,MR)}$
$mn \geq 1$	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$mess\_loss < sn=i, queue = [i^{mn}]>$	$M_{(3b,i),(0,ao,mn-1,an,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$ack\_loss < rn=i,$ $queue = [i^{ao} (i \oplus_{MS} 1)^{an}]>$	$M_{(3b,i),(0,ao-1,mn,an,ret)}^{(MS,MR)}$
$an \geq 1$	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$ack\_loss < rn=i \oplus_{MS} 1,$ $queue = [i^{ao} (i \oplus_{MS} 1)^{an}]>$	$M_{(3b,i),(0,ao,mn,an-1,ret)}^{(MS,MR)}$
$ao \geq 1$	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$receive\_dup\_ack < sn=i, rn=i,$ $queue = [i^{ao-1} (i \oplus_{MS} 1)^{an}]>$	$M_{(3b,i),(0,ao-1,mn,an,ret)}^{(MS,MR)}$
$an \geq 1$	$M_{(3b,i),(0,0,mn,an,ret)}^{(MS,MR)}$	$receive\_ack < sn=i, rn=i \oplus_{MS} 1,$ $rc = ret, queue =$ $[(i \oplus_{MS} 1)^{an-1}]>$	$M_{(4,i \oplus_{MS} 1),(mn,an-1,0,0,0)}^{(MS,MR)}$
none	$M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}$	$send\_ack < rn=i \oplus_{MS} 1,$ $queue = [i^{ao} (i \oplus_{MS} 1)^{an}]>$	$M_{(2b,i),(0,ao,mn,an+1,ret)}^{(MS,MR)}$

Table 8. The set of arcs $A_{(4,i)}^{(MS,MR)}$ with source markings in $V_{(4,i)}^{(MS,MR)}$

Condition	Source Marking	Binding Element	Destination Marking
none	$M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{send\_mess} \langle \text{queue} = [(i \ominus_{MS} 1)^{mo}], \text{sn}=i \rangle$	$M_{(3a,i),(mo,ao,1,0,0)}^{(MS,MR)}$
$mo \geq 1$	$M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{mess\_loss} \langle \text{queue} = [(i \ominus_{MS} 1)^{mo}], \text{sn}=i \ominus_{MS} 1 \rangle$	$M_{(4,i),(mo-1,ao,0,0,0)}^{(MS,MR)}$
$ao \geq 1$	$M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{ack\_loss} \langle \text{queue} = [i^{ao}], \text{rn}=i \rangle$	$M_{(4,i),(mo,ao-1,0,0,0)}^{(MS,MR)}$
$ao \geq 1$	$M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{receive\_dup\_ack} \langle \text{queue} = [i^{ao-1}], \text{sn}=i, \text{rn}=i \rangle$	$M_{(4,i),(mo,ao-1,0,0,0)}^{(MS,MR)}$
none	$M_{(4,i),(mo,ao,0,0,0)}^{(MS,MR)}$	$\text{send\_ack} \langle \text{queue} = [i^{ao}], \text{rn}=i \rangle$	$M_{(1,i),(mo,ao+1,0,0,0)}^{(MS,MR)}$

$A_{(3a,i)}^{(MS,0)}$ and $A_{(4,i)}^{(MS,0)} = \emptyset$ when $MR = 0$. The first column of each arc table gives any additional restrictions that must be placed on the variables mo , ao , mn , an and ret . For example, loss of an old message cannot occur when $mo = 0$.

3 Parametric Language Analysis

In this section we use language equivalence to verify conformance of the SWP to the Stop-and-Wait property of alternating Send and Receive events, i.e. to its service language. The work presented in this section is based on [4, 8], which includes the full proofs.

3.1 The Stop-and-Wait Service Language

The Stop-and-Wait service is only concerned with events visible to the users of the SWP, i.e. sending data at one end and receiving it at the other. Internal protocol mechanisms, such as the use of sequence numbers, acknowledgements and retransmissions are not visible to users. The service language is defined as sequences of the two service primitives, **Send** and **Receive**, as follows:

Definition 4 (Service Language). *The Stop-and-Wait Service Language, $\mathcal{L}_S$, of alternating send and receive events is given by the regular expression $(\text{Send Receive})^* \text{Send}^\dagger$ where $\text{Send}^\dagger$ represents 0 or 1 repetitions of **Send**.*

The service language specifies sequences of alternating send and receive events, which may end with a **Send** or a **Receive** event. When the Stop-and-Wait protocol is operating correctly, one message will be received for every original message sent. Sequences ending with **Send** correspond to the situation (given a lossy medium) where a sender reaches its retransmission limit and gives up. Dealing with this situation is the job of a management entity and so is not reflected in the service language - the sender simply stops.

Recall from the introduction that the protocol language, denoted here as $\mathcal{L}_P$, is all sequences of service primitives exhibited by the protocol. We wish to verify that $\mathcal{L}_P$ conforms to the service language for *all* values of the **MaxSeqNo** and **MaxRetrans** parameters, which we state in the following theorem.

Theorem 2. *The Stop-and-Wait protocol, defined by $CPN_{(MS,MR)}$, conforms to the Stop-and-Wait service, i.e. $\mathcal{L}_P = \mathcal{L}_S = (\text{Send}, \text{Receive})^* \text{Send}^\dagger$, for all values of $MS \geq 1$ and $MR \geq 0$.*

A proof of this theorem is sketched in the rest of this section.

3.2 Obtaining the Parametric Protocol Language

An RG can be interpreted as a Finite State Automaton (FSA) by relabelling arcs with service primitives (or the empty label, ϵ) and defining initial and halt states.

Definition 5 (Mapping Binding Elements to Service Primitives).

Let $\text{Prim} : BE_{(MS,MR)} \rightarrow SP \cup \{\epsilon\}$ be a mapping from the set of binding elements of $CPN_{(MS,MR)}$ to either a service primitive name or to ϵ , where

- $BE_{(MS,MR)}$ is the set of binding elements that occur in $CPN_{(MS,MR)}$; and
- $SP = \{\text{Send}, \text{Receive}\}$ is the set of service primitive names;

such that, for all $(t, b) \in BE_{(MS,MR)}$:

$$\text{Prim}((t, b)) = \begin{cases} \text{Send}, & \text{if } t = \text{send\_mess}, \\ \text{Receive}, & \text{if } t = \text{receive\_mess} \text{ and } sn = rn \text{ holds in } b, \\ \epsilon, & \text{otherwise.} \end{cases}$$

Note that `receive_mess` is only mapped to `Receive` when it corresponds to reception of a new message (i.e. $sn = rn$ in the binding).

The initial state of our parametric FSA is defined as the initial marking of $CPN_{(MS,MR)}$, i.e. $M_0 = M_{(1,0),(0,0,0,0,0)}^{(MS,MR)}$. Because we have an arbitrary number of messages to send from the sender to the receiver, we define a legitimate halt state as any state in which $l \in \mathbb{N}$ messages have been transmitted and successfully acknowledged, so that both the sender and receiver are in their ready states and there are no messages or acknowledgements in the channel. This corresponds to the markings $M_{(1,i),(0,0,0,0,0)}^{(MS,MR)}$ for all $0 \leq i \leq MS$ (incorporating the initial marking). We also include the dead markings of $RG_{(MS,MR)}$ (identified in [6]) in the set of halt states.

Definition 6 (FSA of $RG_{(MS,MR)}$). *The FSA associated with $RG_{(MS,MR)}$ = $(V_{(MS,MR)}, A_{(MS,MR)})$ of $CPN_{(MS,MR)}$, with initial marking M_0 , is $FSA_{RG_{(MS,MR)}} = (V_{(MS,MR)}, SP, \Delta_{(MS,MR)}, M_0, F_{(MS,MR)})$ where*

- $SP = \{\text{Send}, \text{Receive}\}$ is the alphabet;
- $\Delta_{(MS,MR)} = \{M, \text{Prim}((t, b)), M' \mid (M, (t, b), M') \in A_{(MS,MR)}\}$ is the set of transitions labelled by service primitives or epsilons for internal events (the transition relation of the FSA); and
- $F_{(MS,MR)} = \{M_{(1,i),(0,0,0,0,0)}^{(MS,MR)}, M_{(2a,i),(0,0,0,0,MR)}^{(MS,MR)}, M_{(2b,i),(0,0,0,0,MR)}^{(MS,MR)} \mid 0 \leq i \leq MS\}$ is the set of final states.

The states of $FSA_{RG(MS,MR)}$ are the nodes of $RG(MS,MR)$, given by Table 2. The arcs of $FSA_{RG(MS,MR)}$ are given by Tables 3 to 8 by applying *Prim* to each binding element (arc label). In the remainder of this section, we will often refer to nodes and edges in $FSA_{RG(MS,MR)}$ by the corresponding markings and arcs in $RG(MS,MR)$. This provides a way of easily identifying the nodes of $FSA_{RG(MS,MR)}$ and the edges whose labels are mapped to ϵ by *Prim*.

3.3 Epsilon Closures

Before proceeding with FSA reduction we calculate symbolically the ϵ -closure [9] of any (arbitrary) marking in $V_{(2a,i)}^{(MS,MR)}$ and $V_{(3b,i)}^{(MS,MR)}$ of Table 2. Symbolic expressions for the ϵ -closure of class 1, 2b, 3a or 4 markings are not needed, due to the lazy subset evaluation technique for determinisation [9] used in Section 3.4. In addition, we develop expressions that specify the set of non- ϵ -labelled outgoing edges from the markings in these ϵ -closures. These expressions are useful in the determinisation procedure. We define the ϵ -closure operation based on the narrative description in [9].

Definition 7 (ϵ -closure)

For $FSA = (S, \Sigma, \Delta, s_0, F)$ the ϵ -closure of $s \in S$ is given by $Closure : S \rightarrow 2^S$, where $Closure(s) = \{s' \mid s \xrightarrow{\epsilon^*} s'\}$ and $s \xrightarrow{\epsilon^*} s'$ indicates that s' can be reached from s via a contiguous sequence of 0 or more ϵ moves.

Note that $s \in Closure(s)$.

The ϵ -closure of Class 2a nodes

Lemma 1. The ϵ -closure of a marking, $M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)} \in V_{(2a,i)}^{(MS,MR)}$, is given by:

$$\begin{aligned}
 & Closure(M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}) \\
 &= \{M_{(2a,i),(mo',ao',mn',0,ret')}^{(MS,MR)} \mid mo' + ao' \leq mo + ao, 0 \leq mo' \leq mo, \\
 &\quad mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \\
 &\cup \{M_{(3a,i),(mo',ao',mn',0,ret')}^{(MS,MR)} \mid mo' + ao' \leq mo + ao - 1, 0 \leq mo' \leq mo - 1, \\
 &\quad mn' \leq mn + ret' - ret, ret \leq ret' \leq MR, mo > 0\}
 \end{aligned} \tag{1}$$

Proof Sketch. In [8] this result is presented as a corollary to a longer derivation. In this paper we present only the intuition behind its correctness. For full details of the derivation see [8].

The class 2a markings in the above closure come about in the following way. If old messages exist (i.e. $mo > 0$) an old message can be received by the receiver (an ϵ -transition, row 4 of Table 4) and acknowledged (also an ϵ -transition, row 6 of Table 6). The result is one fewer old messages and one additional old acknowledgement. This can occur for all of the old messages, so that $0 \leq mo' \leq mo$ and $mo' + ao' = mo + ao$ describe the number of old messages and acknowledgements

and captures the relationship between them. Given the arbitrary loss property of the channels, any of the old messages or acknowledgements can be lost (also an ϵ -transition), resulting in $mo' + ao' \leq mo + ao, 0 \leq mo' \leq mo$.

In terms of the new messages, mn' , row 1 of Table 4 (the `timeout_retrans` ϵ -transition) indicates that the new message can be retransmitted until the retransmission limit is reached (i.e. until $ret = MR$). This allows at most $MR - ret$ further retransmissions, each time incrementing the number of new messages, mn' , and the retransmission counter, ret' . Given that we started with mn new messages, the result is $mn' = mn + (ret' - ret), ret \leq ret' \leq MR$, where $ret' - ret$ is the number of additional retransmissions of the new message that have occurred. Again, due to the arbitrary loss property of our channel, the resulting expression is $mn' \leq mn + ret' - ret, ret \leq ret' \leq MR$.

The class 3a markings in the above closure have a very similar form to the class 2a markings. The only differences in the inequalities are the two additional restrictions: $mo' \leq mo - 1$ and $mo > 0$. This is because class 3a markings are reached from class 2a markings by an old message being received by the receiver. Hence, to reach a class 3a marking, at least one old message must exist (i.e. $mo > 0$) and consequently there will be one fewer old messages in the message channel when it is received (i.e. $0 \leq mo' \leq mo - 1$).

The fact that the closure given above covers all markings reachable from $M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$ via zero or more ϵ -transitions is evident from the arc tables. All rows in Tables 4 and 6 except for row 5 of Table 4 correspond to transitions that result in either a class 2a or 3a marking. Detailed analysis of each [8] reveals that all of these destination markings fall within the closure defined in the lemma. $M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$ itself is included in the closure, hence precisely those markings reachable from $M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}$ via 0 or more ϵ -transitions are specified in the closure and the lemma is proved. $\square$

Row 5 of Table 4 maps to the `Receive` service primitive, not to ϵ . From row 5 of Table 4, the only non- ϵ -labelled outgoing edges from the closure given in Lemma 1 are given by:

$$\begin{aligned} & OutEdges(Closure(M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)})) \\ &= \{(M_{(2a,i),(0,ao',mn',0,ret')}^{(MS,MR)}, \text{Receive}, M_{(3b,i),(0,ao',mn'-1,0,ret')}^{(MS,MR)}) \mid M_{(2a,i),(0,ao',mn',0,ret')}^{(MS,MR)} \\ &\quad \in Closure(M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)}), mn' > 0\} \end{aligned} \quad (2)$$

Substituting the relevant part of the closure (the class 2a markings) from Equation (1) into Equation (2) we obtain:

$$\begin{aligned} & OutEdges(Closure(M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)})) \\ &= \{(M_{(2a,i),(0,ao',mn',0,ret')}^{(MS,MR)}, \text{Receive}, M_{(3b,i),(0,ao',mn'-1,0,ret')}^{(MS,MR)}) \mid ao' \leq mo + ao, \\ &\quad 0 < mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \end{aligned} \quad (3)$$

The ϵ -closure of Nodes in $V_{(3b,i)}^{(MS,MR)}$

Lemma 2. *The ϵ -closure of a marking, $M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)} \in V_{(3b,i)}^{(MS,MR)}$, is given by:*

$$\begin{aligned}
 & Closure(M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)}) \\
 &= \{M_{(3b,i),(0,ao',mn',an',ret')}^{(MS,MR)} \mid ao' \leq ao, mn' + an' \leq mn + an + ret' - ret, \\
 &\quad 0 \leq mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \\
 &\cup \{M_{(2b,i),(0,ao',mn',an',ret')}^{(MS,MR)} \mid ao' \leq ao, mn' + an' \leq mn + an + ret' - ret + 1, \\
 &\quad 0 \leq mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \\
 &\cup \{M_{(4,i \oplus_{MS} 1),(mo',ao',0,0,0)}^{(MS,MR)} \mid mo' + ao' \leq mn + an + ret' - ret - 1, \\
 &\quad 0 \leq mo' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \\
 &\cup \{M_{(1,i \oplus_{MS} 1),(mo',ao',0,0,0)}^{(MS,MR)} \mid mo' + ao' \leq mn + an + ret' - ret, \\
 &\quad 0 \leq mo' \leq mn + ret' - ret, ret \leq ret' \leq MR\}
 \end{aligned} \tag{4}$$

Proof Sketch. The proof follows a similar procedure to that of Lemma 1. It is complicated by the fact that, from a class 3b marking, it is possible to reach markings of class 3b, class 2b, class 1 and class 4, via 0 or more ϵ -transitions. Furthermore, the sender sequence number for the class 1 and 4 markings has been incremented, i.e. is one greater (modulo $\text{MaxSeqNo} + 1$) than the class 3b marking to whose ϵ -closure they belong. For brevity, we omit the details of this proof here, and refer the reader to [8]. $\square$

Row 1 of both Table 3 and Table 8 define arcs that map to the **Send** primitive, leading from a class 1 or 4 marking to a class 2a or 3a marking respectively. Omitting the details (see [8]), the set of non- ϵ -labelled outgoing edges is obtained in a similar way to those in Equation (3), giving:

$$\begin{aligned}
 & OutEdges(Closure(M_{(3b,i),(0,ao,mn,an,ret)}^{(MS,MR)})) \\
 &= \{(M_{(4,i \oplus_{MS} 1),(mn',an',0,0,0)}^{(MS,MR)}, \text{Send}, M_{(3a,i \oplus_{MS} 1),(mn',an',1,0,0)}^{(MS,MR)}) \mid \\
 &\quad mn' + an' \leq mn + an + ret' - ret - 1, 0 \leq mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\} \\
 &\cup \{(M_{(1,i \oplus_{MS} 1),(mn',an',0,0,0)}^{(MS,MR)}, \text{Send}, M_{(2a,i \oplus_{MS} 1),(mn',an',1,0,0)}^{(MS,MR)}) \mid \\
 &\quad mn' + an' \leq mn + an + ret' - ret, 0 \leq mn' \leq mn + ret' - ret, ret \leq ret' \leq MR\}
 \end{aligned} \tag{5}$$

3.4 Determinisation

Now that we have expressions for the ϵ -closures we require, determinisation of $FSA_{RG(MS,MR)}$, using lazy subset construction, proceeds as described in [9].

We define a deterministic FSA as $DFSA = (S^{det}, \Sigma, \Delta^{det}, s_0^{det}, F^{det})$. The initial state of $DFSA_{RG(MS,MR)}$, which we denote C_0 , is the ϵ -closure of the initial state of $FSA_{RG(MS,MR)}$. From inspection of Table 3:

$$C_0 = s_0^{det} = Closure(s_0) = Closure(M_{(1,0),(0,0,0,0,0)}^{(MS,MR)}) = \{M_{(1,0),(0,0,0,0,0)}^{(MS,MR)}\} \tag{6}$$

as there are no outgoing ϵ edges from the initial state of $FSA_{RG(MS,MR)}$ regardless of the values of MS and MR . Furthermore, $C_0 \in S^{det}$.

The only transition that is enabled by the single state in C_0 (i.e. the initial marking of the CPN) is **send_mess** (row 1 of Table 3 where $i = 0$) which maps to the **Send** service primitive. The destination marking is $M_{(2a,0),(0,0,1,0,0)}^{(MS,MR)}$. The ϵ -closure of $M_{(2a,0),(0,0,1,0,0)}^{(MS,MR)}$, which we denote C_1 , is given by evaluating Equation (1) for $M_{(2a,0),(0,0,1,0,0)}^{(MS,MR)}$:

$$\begin{aligned} C_1 &= Closure(M_{(2a,0),(0,0,1,0,0)}^{(MS,MR)}) \\ &= \{M_{(2a,0),(mo,ao,mn,0,ret)}^{(MS,MR)} \mid mo+ao \leq 0, 0 \leq mo \leq 0, 0 \leq mn \leq 1+ret, 0 \leq ret \leq MR\} \\ &= \{M_{(2a,0),(0,0,mn,0,ret)}^{(MS,MR)} \mid 0 \leq mn \leq 1+ret, 0 \leq ret \leq MR\} \end{aligned} \quad (7)$$

Thus $C_1 \in S^{det}$ and $(s_0^{det}, \text{Send}, C_1) \in \Delta^{det}$.

C_0 and C_1 are illustrated in Fig. 2. C_0 is shown as a large red circle within $V_{(1,0)}^{(MS,MR)}$, which leads to the successor $C_1 \in V_{(2a,0)}^{(MS,MR)}$ via the **Send** primitive. There are no class 3a markings in C_1 and only some of the markings in $V_{(2a,0)}^{(MS,MR)}$ are covered, hence C_1 has been depicted to reflect this.

To obtain the successors of nodes in the closure, C_1 , reachable via non- ϵ moves, we substitute C_1 into Equation (3) to obtain the set of outgoing non- ϵ arcs:

$$\begin{aligned} OutEdges(C_1) &= OutEdges(Closure(M_{(2a,0),(0,0,1,0,0)}^{(MS,MR)})) = \{(M_{(2a,0),(0,ao,mn,0,ret)}^{(MS,MR)}, \\ &\quad \text{Receive}, M_{(3b,0),(0,ao,mn-1,0,ret)}^{(MS,MR)}) \mid ao \leq 0, 0 < mn \leq 1+ret, 0 \leq ret \leq MR\} \end{aligned} \quad (8)$$

Thus the direct successors of markings in C_1 , via the **Receive** primitive, are the destination nodes from Equation (8):

$$DirectSucc_{C_1} = \{M_{(3b,0),(0,0,mn-1,0,ret)}^{(MS,MR)} \mid 0 < mn \leq 1+ret, 0 \leq ret \leq MR\} \quad (9)$$

The successor of C_1 is the union of the ϵ -closure of all nodes in $DirectSucc_{C_1}$. However, rather than calculate the ϵ -closure of every node in $DirectSucc_{C_1}$, let us consider the ϵ -closure of just one, $M_{(3b,0),(0,0,0,0,0)}^{(MS,MR)}$, where $mn - 1 = ret = 0$. Substituting this into Equation (4) and simplifying to remove redundant inequalities (see [8]) gives:

$$\begin{aligned} C_2 &= \{M_{(3b,0),(0,0,mn,an,ret)}^{(MS,MR)} \mid 0 \leq mn+an \leq ret, 0 \leq ret \leq MR\} \\ &\cup \{M_{(2b,0),(0,0,mn,an,ret)}^{(MS,MR)} \mid 0 \leq mn+an \leq ret+1, 0 \leq mn \leq ret, 0 \leq ret \leq MR\} \\ &\cup \{M_{(4,1),(mo,ao,0,0,0)}^{(MS,MR)} \mid 0 \leq mo+ao \leq MR-1\} \\ &\cup \{M_{(1,1),(mo,ao,0,0,0)}^{(MS,MR)} \mid 0 \leq mo+ao \leq MR\} \end{aligned} \quad (10)$$

In [8] we show that $C_2 = \bigcup_{M \in DirectSucc_{C_1}} Closure(M)$, i.e. that C_2 is actually the union of the ϵ -closure of all markings in $DirectSucc_{C_1}$.

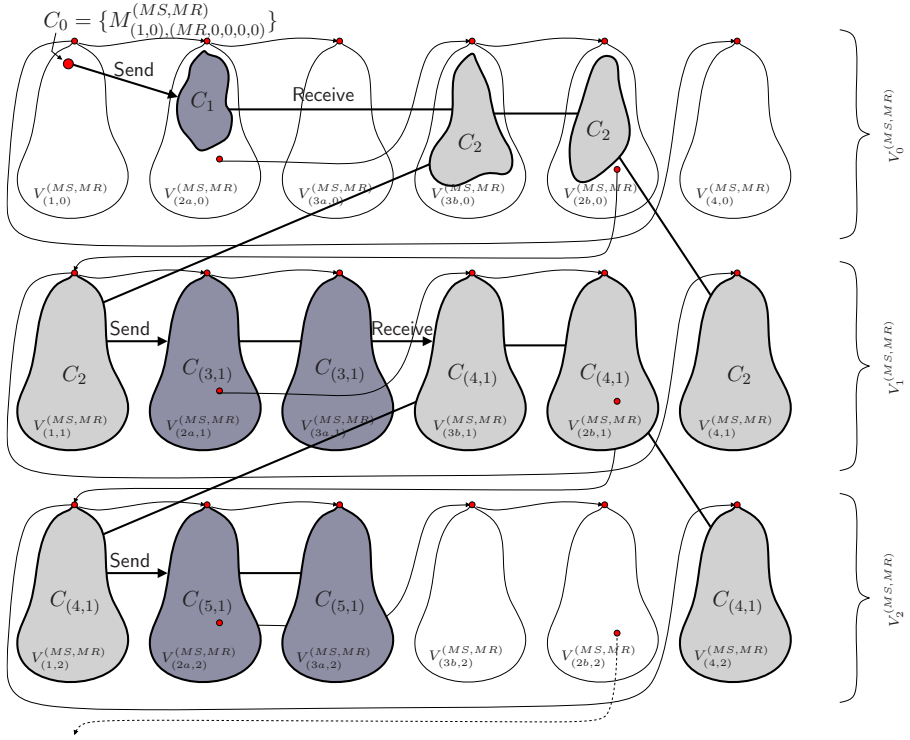


Fig. 2. Construction of a deterministic FSA showing the closures, C_0 , C_1 , C_2 , $C_{(3,1)}$, $C_{(4,1)}$ and $C_{(5,1)}$, and the arcs between them

Lemma 3. $C_2 = \bigcup_{M \in \text{DirectSucc}_{C_1}} \text{Closure}(M)$.

Proof. See [8] for details. $\square$

Corollary 1. $C_2 \in S^{\text{det}}$ and $(C_1, \text{Receive}, C_2) \in \Delta^{\text{det}}$

This result is also illustrated in Fig. 2. Note that although the set C_2 comprises states from $V_{(2b,0)}^{(MS,MR)}$, $V_{(3b,0)}^{(MS,MR)}$, $V_{(1,1)}^{(MS,MR)}$ and $V_{(4,1)}^{(MS,MR)}$, the states in all four of these subsets in Fig. 2 are part of the single successor state of C_1 . Hence, in Fig. 2, the four shaded subsets representing C_2 are connected via solid lines with no arrowheads.

The destination markings of non- ϵ edges originating in C_2 are obtained by following the same procedure as previously used. Omitting details (see [8]) we obtain:

$$\begin{aligned} \text{DirectSucc}_{C_2} = & \{M_{(3a,1),(mn,an,1,0,0)}^{(MS,MR)} \mid 0 \leq mn + an \leq MR - 1\} \\ & \cup \{M_{(2a,1),(mn,an,1,0,0)}^{(MS,MR)} \mid 0 \leq mn + an \leq MR\} \end{aligned} \quad (11)$$

Note that $MR > 0$ is implicit in the set defining class 3a markings above.

Now, the successor of C_2 is the union of the ϵ -closure of each marking in $DirectSucc_{C_2}$. Let us leave the concrete domain at this point. We know from Equation (10) that C_2 spans *all* markings in $V_{(1,1)}^{(MS,MR)}$ and $V_{(4,1)}^{(MS,MR)}$. Consider the set of states obtained by replacing the sender sequence number, 1, with i , in Equation (11):

$$DirectSucc_{(C_2,i)} = \{M_{(3a,i),(mo,ao,1,0,0)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR - 1\} \\ \cup \{M_{(2a,i),(mo,ao,1,0,0)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR\} \quad (12)$$

for each $i, 0 \leq i \leq MS$. This gives a family of sets of markings, each identical to $DirectSucc_{C_2}$ apart from a uniform translation of sequence numbers. When $i = 1$ we have $DirectSucc_{(C_2,1)} = DirectSucc_{C_2}$.

To compute the union of the ϵ -closure of all nodes in $DirectSucc_{(C_2,i)}$ we employ a similar procedure to that for discovering C_2 . We select $M_{(2a,i),(MR,0,1,0,0)}^{(MS,MR)} \in DirectSucc_{(C_2,i)}$ and represent its ϵ -closure by $C_{(3,i)}$, where the appearance of i in the subscript shows that $C_{(3,i)}$ is parametric. Omitting details of its derivation (see [8]) $C_{(3,i)}$ is given by:

$$C_{(3,i)} = \{M_{(2a,i),(mo,ao,mn,0,ret)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR, 0 \leq mn \leq 1 + ret, 0 \leq ret \leq MR\} \\ \cup \{M_{(3a,i),(mo,ao,mn,0,ret)}^{(MS,MR)} \mid 0 \leq mo + ao \leq MR - 1, 0 \leq mn \leq 1 + ret, 0 \leq ret \leq MR\} \quad (13)$$

By inspection of Table 2, $C_{(3,i)}$ equals exactly the set of all class 2a and 3a nodes for a given i , i.e.

$$C_{(3,i)} = V_{(2a,i)}^{(MS,MR)} \cup V_{(3a,i)}^{(MS,MR)} \quad (14)$$

$C_{(3,i)}$ is equal to the union of the ϵ -closure of all markings in $DirectSucc_{(C_2,i)}$:

Lemma 4. $C_{(3,i)} = \bigcup_{M \in DirectSucc_{(C_2,i)}} Closure(M)$.

Proof. See [8] for details. $\square$

Corollary 2. When $i = 1$, $C_{(3,1)} \in S^{det}$ and $(C_2, Send, C_{(3,1)}) \in \Delta^{det}$.

We illustrate this result in Fig. 2, which shows $C_{(3,1)}$ covering all markings in $V_{(2a,1)}^{(MS,MR)}$ and $V_{(3a,1)}^{(MS,MR)}$, and the arc from C_2 to $C_{(3,1)}$ labelled with the *Send* service primitive.

To obtain the destination markings of non- ϵ edges originating in $C_{(3,i)}$ we follow the same procedure as for discovering the outgoing non- ϵ edges originating in C_2 . We denote this set of destination markings by $DirectSucc_{(C_{3,i})}$. We then obtain the union of the ϵ -closure of all of these destination markings in the same way as above, by finding the ϵ -closure of a carefully selected marking and proving that it equals the union of the ϵ -closures of all of the destination markings. Omitting details (see [8]) we obtain:

$$C_{(4,i)} = V_{(2b,i)}^{(MS,MR)} \cup V_{(3b,i)}^{(MS,MR)} \cup V_{(1,i \oplus_{MS} 1)}^{(MS,MR)} \cup V_{(4,i \oplus_{MS} 1)}^{(MS,MR)} \quad (15)$$

Lemma 5. $C_{(4,i)} = \bigcup_{M \in \text{DirectSucc}_{(C_{3,i})}} \text{Closure}(M)$.

Proof. See [8] for details. $\square$

Corollary 3. When $i = 1$, $C_{(4,1)} \in S^{det}$ and hence $(C_{(3,1)}, \text{Receive}, C_{(4,1)}) \in \Delta^{det}$.

This is also illustrated in Fig. 2. Unlike C_2 , $C_{(4,1)}$ covers all class 2b and class 3b markings in $V_1^{(MS,MR)}$, whereas C_2 only covers some of the class 2b and class 3b markings in $V_0^{(MS,MR)}$. It is important to note, for when we construct our parametric deterministic FSA, that $C_{(4,0)}$ (obtained by substituting $i = 0$ into Equation (15)) and C_2 are not equal.

We repeat the process to find the destination markings of the outgoing non- ϵ -labelled edges from the markings in $C_{(4,i)}$, which we denote $\text{DirectSucc}_{(C_{4,i})}$. From this, we obtain the successor state of $C_{(4,i)}$, which we denote $C_{(5,i)}$, and which is given by:

$$C_{(5,i)} = V_{(2a,i \oplus_{MS} 1)}^{(MS,MR)} \cup V_{(3a,i \oplus_{MS} 1)}^{(MS,MR)} \quad (16)$$

By inspection of Equations (13) and (16), we find that $C_{(5,i)}$ is equal to $C_{(3,i \oplus_{MS} 1)}$.

Lemma 6. $C_{(3,i \oplus_{MS} 1)} = \bigcup_{M \in \text{DirectSucc}_{(C_{4,i})}} \text{Closure}(M)$.

Proof. See [8] for details. $\square$

Corollary 4. When $i = 1$, $C_{(3,2)} \in S^{det}$ and hence $(C_{(4,1)}, \text{Send}, C_{(3,2)}) \in \Delta^{det}$.

This result is also illustrated in Fig. 2 for $C_{(5,1)}$. Note that $C_{(5,1)}$ ($C_{(3,2)}$) covers all class 2a and class 3a markings in $V_2^{(MS,MR)}$, and that $C_{(3,1)}$ covers all class 2a and class 3a markings in $V_1^{(MS,MR)}$. We can now state and prove a lemma that builds the rest of the structure of our deterministic parametric FSA through lazy subset evaluation:

Lemma 7. $\forall i \in \{0, 1, \dots, MS\}$, $C_{(3,i)} \in S^{det}$, $C_{(4,i)} \in S^{det}$, $(C_{(3,i)}, \text{Receive}, C_{(4,i)}) \in \Delta^{det}$, and $(C_{(4,i)}, \text{Send}, C_{(3,i \oplus_{MS} 1)}) \in \Delta^{det}$.

Proof. We know from direct construction and Corollary 1 that $C_2 \in S^{det}$ and from Corollary 2 that when $i = 1$ is substituted into Equation (13), we get $C_{(3,1)} \in S^{det}$ and $(C_2, \text{Send}, C_{(3,1)}) \in \Delta^{det}$. From Corollary 3, we know that when substituting $i = 1$ into Equation (15), we get $C_{(4,1)} \in S^{det}$ and $(C_{(3,1)}, \text{Receive}, C_{(4,1)}) \in \Delta^{det}$. We know that when substituting $i = 1$ into Equation (16) and $i = 2$ into Equation (13) we get $C_{(5,1)} = C_{(3,2)}$. Hence, from Corollary 4, $C_{(3,2)} \in S^{det}$ and $(C_{(4,1)}, \text{Send}, C_{(3,2)}) \in \Delta^{det}$. Repeating the application of Corollaries 3 and 4 for $i = 2, 3, \dots, MS$ we get $\{C_{(4,i)}, C_{(3,i \oplus_{MS} 1)} \mid 2 \leq i \leq MS\} \subset S^{det}$ and $\{(C_{(3,i)}, \text{Receive}, C_{(4,i)}), (C_{(4,i)}, \text{Send}, C_{(3,i \oplus_{MS} 1)}) \mid 2 \leq i \leq MS\} \subset \Delta^{det}$. Finally, from Corollary 3 when $i = 0$ we obtain $C_{(4,0)} \in S^{det}$ and $(C_{(3,0)}, \text{Receive}, C_{(4,0)}) \in \Delta^{det}$, and from Corollary 4 when $i = 0$, $(C_{(4,0)}, \text{Send}, C_{(3,1)}) \in \Delta^{det}$. $\square$

Lemma 7, along with C_0 and $C_1 \in S^{det}$ and $(C_0, \text{Send}, C_1), (C_1, \text{Receive}, C_2) \in \Delta^{det}$, means we have explored all states in S^{det} and all outgoing arcs of states in S^{det} , starting from the initial state, using lazy evaluation. All that remains to complete $DFSA_{RG(MS, MR)}$ is identification of halt states.

According to [9], halt states of $DFSA_{RG(MS, MR)}$ are those subsets of states of $FSA_{RG(MS, MR)}$ that contain halt states of $FSA_{RG(MS, MR)}$. The halt states of $FSA_{RG(MS, MR)}$ are given in Definition 6. From Equation (6) s_0^{det} is trivially a halt state. From Equations (7) and (13) C_1 and $C_{(3,i)}$ (for each $i \in \{0, 1, \dots, MS\}$) are halt states because they contain $M_{(2a,0),(0,0,0,0,MR)}^{(MS,MR)}$. From Equations (10) and (15) C_2 and $C_{(4,i)}$ (for each $i \in \{0, 1, \dots, MS\}$) are halt states because they contain $M_{(2b,0),(0,0,0,0,MR)}^{(MS,MR)}$. Thus all states of $DFSA_{RG(MS, MR)}$ are halt states.

$DFSA_{RG(MS, MR)}$ is thus given by

$$DFSA_{RG(MS, MR)} = (S_{(MS, MR)}^{det}, SP, \Delta_{(MS, MR)}^{det}, s_0^{det}, F_{(MS, MR)}^{det})$$

where:

- $S_{(MS, MR)}^{det} = \{C_0, C_1, C_2\} \cup \{C_{(3,i)}, C_{(4,i)} \mid 0 \leq i \leq MS\}$;
- $\Delta_{(MS, MR)}^{det} = \{(C_0, \text{Send}, C_1), (C_1, \text{Receive}, C_2), (C_2, \text{Send}, C_{(3,1)})\} \cup \{(C_{(3,i)}, \text{Receive}, C_{(4,i)}), (C_{(4,i)}, \text{Send}, C_{(3,i \oplus MS1)}) \mid 0 \leq i \leq MS\}$;
- $s_0^{det} = \text{Closure}(M_{(1,0),(0,0,0,0,0)}^{(MS,MR)}) = C_0$; and
- $F_{(MS, MR)}^{det} = S_{(MS, MR)}^{det}$.

$DFSA_{RG(MS, MR)}$ is represented in tabular form in Table 9 and graphically in Fig. 3. By convention, the initial state is shown highlighted in bold and the halt states (all states) are shown as double circles in Fig. 3. The main loop in the lower half of the figure illustrates the repeated behaviour over all values of i , $0 \leq i \leq MS$, of alternating Send and Receive events, moving from $C_{(3,i)}$ to $C_{(4,i)}$ on a Receive event and from $C_{(4,i)}$ to $C_{(3,i \oplus MS1)}$ on a Send event.

3.5 Minimisation and Conformance to the SWP Service Language

Determinisation has removed the effect of the MaxRetrans parameter on the parametric FSA. However, the deterministic FSA representing the protocol language is not minimal. This is evident from Fig. 3, which represents the language generated by the regular expression $(\text{Send}, \text{Receive})^* \text{Send}^\dagger$, but which could be represented by a FSA with fewer states.

Table 9. $DFSA_{RG(MS, MR)}$, where rows 4 and 5 are evaluated for $0 \leq i \leq MS$

Source node	Arc Label	Dest. node	Dest. = Halt?
C_0	Send	C_1	true
C_1	Receive	C_2	true
C_2	Send	$C_{(3,1)}$	true
$C_{(3,i)}$	Receive	$C_{(4,i)}$	true
$C_{(4,i)}$	Send	$C_{(3,i \oplus MS1)}$	true

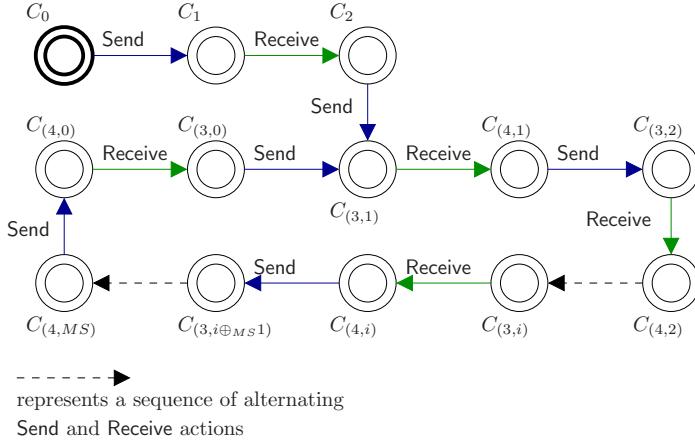


Fig. 3. An abstract visualisation of the parametric deterministic FSA, $DFSARG_{(MS,MR)}$

Following the minimisation procedure described in [1], from $DFSARG_{(MS,MR)}$ (and Table 9) it can be seen that all states are halt states, so we begin with all states placed in the same subset, i.e. $\{C_0, C_1, C_2, C_{(3,i)}, C_{(4,i)} \mid 0 \leq i \leq MS\}$. States are now separated into disjoint subsets based on the input symbols they accept and the subset to which the resulting successor states belong. This results in the subset $Set1 = \{C_0, C_2, C_{(4,i)} \mid 0 \leq i \leq MS\}$ of states accepting only the input symbol **Send**, and the subset $Set2 = \{C_1, C_{(3,i)} \mid 0 \leq i \leq MS\}$ of states accepting only the input symbol **Receive**. These subsets cannot be further divided in this way, as all states in $Set1$ accept only **Send**, leading to a successor in $Set2$, and all states in $Set2$ accept only **Receive**, leading to a successor in $Set1$. These two subsets become, essentially, the states of the minimised FSA. We choose the representative ‘1’ to represent $Set1$ in the minimal FSA and the representative ‘2’ to represent $Set2$. Both are halt states and ‘1’ is the initial state, as $Set1$ contains the initial state of the deterministic FSA, C_0 . **Send** and **Receive** edges are defined accordingly. The resulting minimised deterministic FSA is given by $MFSARG_{(MS,MR)} = (S_{(MS,MR)}^{min}, SP, \Delta_{(MS,MR)}^{min}, 1, F_{(MS,MR)}^{min})$ where:

- $S_{(MS,MR)}^{min} = \{1, 2\}$;
- $\Delta_{(MS,MR)}^{min} = \{(1, \text{Send}, 2), (2, \text{Receive}, 1)\}$; and
- $F_{(MS,MR)}^{min} = S_{(MS,MR)}^{min} = \{1, 2\}$.

Note that this FSA is completely independent of the values of the parameters MS and MR . Determinisation removes events that are invisible to the user, such as retransmissions. Hence, it is not unexpected that determinisation has removed the effect of MR . Minimisation collapses repeated behaviour, such as the (**Send** **Receive**) event sequence that is repeated for each sequence number. Thus, it makes sense intuitively that the effect of MS is removed by minimisation.

$MFS_{RG(MS,MR)}$ thus represents the protocol language for *all* of the members of the infinite family of Stop-and-Wait protocol models, i.e. *all* positive values of MS and *all* non-negative values of MR . The language of $MFS_{RG(MS,MR)}$ is identical to the Stop-and-Wait service of $(\text{Send}, \text{Receive})^* \text{Send}^\dagger$. This verifies that the SWP does indeed satisfy the Stop-and-Wait property for all allowable values of the parameters, and thus Theorem 2 is proved.

4 Conclusions and Future Work

In this paper we have presented a further step in the parametric verification of the infinite class of Stop-and-Wait Protocols, that of language analysis. We have successfully verified that all instantiations of the Stop-and-Wait Protocol (abstracting from data) operating over an in-order but lossy medium, parameterised with the maximum sequence number and maximum number of retransmissions, conform to the Stop-and-Wait service language. This was accomplished by deriving a parametric FSA directly from the parametric reachability graph, then applying automata reduction techniques directly to the parametric FSA to reveal a simple, non-parametric FSA describing the service language.

The Stop-and-Wait Protocol class is important because it represents an infinite class of transport protocols in two parameters. Its verification provides the first step in generalising the approach to credit-based flow control protocols, which include widely used protocols such as the Internet's Transmission Control Protocol. Credit-based flow control protocols have three parameters (the third is the maximum size of a credit-based window). We have taken the first steps in extending this work to credit-based flow control protocols by considering both batch and first-only retransmission schemes and in-order and in-window accept policies of the receiver in [3], where we derive expressions in the parameters for the number of terminal markings and the channel bounds. We would also like to extend this work to complete the verification of the Stop-and-Wait class over lossy in-order channels, by incorporating data independence principles [13, 14] so that data properties (such as data is never lost or duplicated) can be verified. Finally we would like to consider if a similar approach can be used to verify the Stop-and-Wait class operating over a re-ordering and lossy medium, as is possible over the Internet.

References

1. Barrett, W.A., Bates, R.M., Gustafson, D.A., Couch, J.D.: Compiler Construction: Theory and Practice. 2nd edn. Science Research Associates (1986)
2. Billington, J., Gallasch, G.E., Han, B.: A Coloured Petri Net Approach to Protocol Verification. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 210–290. Springer, Heidelberg (2004)
3. Billington, J., Saboo, S.: An investigation of credit-based flow control protocols. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008), Marseille, France, 3-7 March 2008, 10 pages (2008) (CD ROM), ACM International Conference Proceedings series (to appear)

4. Gallasch, G.E.: Parametric Verification of the Class of Stop-and-Wait Protocols. PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia (May 2007)
5. Gallasch, G.E., Billington, J.: Using Parametric Automata for the Verification of the Stop-and-Wait Class of Protocols. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 457–473. Springer, Heidelberg (2005)
6. Gallasch, G.E., Billington, J.: A Parametric State Space for the Analysis of the Infinite Class of Stop-and-Wait Protocols. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 201–218. Springer, Heidelberg (2006)
7. Gallasch, G.E., Billington, J.: Parametric Verification of the Class of Stop-and-Wait Protocols over Ordered Channels. Technical Report CSEC-23, Computer Systems Engineering Centre Report Series, University of South Australia (2006)
8. Gallasch, G.E., Billington, J.: Language Analysis of the Class of Stop-and-Wait Protocols. Technical Report CSEC-31, Computer Systems Engineering Centre Report Series, University of South Australia (2008)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. 2nd edn. Addison-Wesley, Reading (2001)
10. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1: Basic Concepts. 2nd edn. Monographs in Theoretical Computer Science, Springer, Heidelberg (1997)
11. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 9(3-4), 213–254 (2007)
12. Mohri, M.: Generic Epsilon-Removal and Input Epsilon-Normalisation Algorithms for Weighted Transducers. International Journal of Foundations of Computer Science 13(1), 129–143 (2002)
13. Sabnani, K.: An Algorithmic Technique for Protocol Verification. IEEE Transactions on Communications 36(8), 924–931 (1988)
14. Wolper, P.: Expressing Interesting Properties of Programs in Propositional Temporal Logic. In: Proceedings of the 13th Annual ACM Symposium on Principles of Programming Languages (POPL), pp. 184–193. ACM Press, New York (1986)

Hierarchical Set Decision Diagrams and Automatic Saturation[★]

Alexandre Hamez^{1,2}, Yann Thierry-Mieg¹, and Fabrice Kordon¹

¹ Université P. & M. Curie

LIP6 - CNRS UMR 7606

4 Place Jussieu, 75252 Paris cedex 05, France

² EPITA

Research and Development Laboratory

F-94276 Le Kremlin-Bicetre cedex, France

Alexandre.Hamez@lip6.fr, Yann.Thierry-Mieg@lip6.fr,

Fabrice.Kordon@lip6.fr

Abstract. Shared decision diagram representations of a state-space have been shown to provide efficient solutions for model-checking of large systems. However, decision diagram manipulation is tricky, as the construction procedure is liable to produce intractable intermediate structures (a.k.a peak effect). The definition of the so-called saturation method has empirically been shown to mostly avoid this peak effect, and allows verification of much larger systems. However, applying this algorithm currently requires deep knowledge of the decision diagram data-structures, of the model or formalism manipulated, and a level of interaction that is not offered by the API of public DD packages.

Hierarchical Set Decision Diagrams (SDD) are decision diagrams in which arcs of the structure are labeled with sets, themselves stored as SDD. This data structure offers an elegant and very efficient way of encoding structured specifications using decision diagram technology. It also offers, through the concept of inductive homomorphisms, unprecedented freedom to the user when defining the transition relation. Finally, with very limited user input, the SDD library is able to optimize evaluation of a transition relation to produce a saturation effect at runtime. We further show that using recursive folding, SDD are able to offer solutions in logarithmic complexity with respect to other DD. We conclude with some performances on well known examples.

Keywords: Hierarchical Decision Diagrams, Model Checking, Saturation.

1 Introduction

Parallel systems are notably difficult to verify due to their complexity. Non-determinism of the interleaving of elementary actions in particular is a source of errors difficult to detect through testing. Model-checking of finite systems or exhaustive exploration of the state-space is very simple in its principle, entirely automatic, and provides useful counter-examples when the desired property is not verified.

[★] This work has been partially supported by the ModelPlex European integrated project FP6-IP 034081 (Modeling Solutions for Complex Systems).

However model-checking suffers from the combinatorial state-space explosion problem, that severely limits the size of systems that can be checked automatically. One solution which has shown its strength to tackle very large state spaces is the use of shared decision diagrams like BDD [1,2].

But decision diagram technology also suffers from two main drawbacks. First, the order of variables has a huge impact on performance and defining an appropriate order is non-trivial [3]. Second, the way the transition relation is defined and applied may have a huge impact on performance [4,5].

The objective of this paper is to present novel optimization techniques for hierarchical decision diagrams called Set Decision Diagrams (SDD), suitable to master the complexity of very large systems. Although SDD are a general all-purpose compact data-structure, a design goal has been to provide easy to use off the shelf constructs (such as a fixpoint) to develop a model-checker using SDD. These constructs allow the library to control operation application, and harness the power of state of the art saturation algorithms [5] with low user expertise in DD.

No specific hypothesis is made on the input language, although we focus here on a system described as a composition of labeled transition systems. This simple formalism captures most transition-based representations (such as automata, communicating processes like in Promela [6], Harel state charts, or bounded Petri nets).

Our hierarchical Set Decision Diagrams (section 2) offer the following capabilities:

- Using the structure of a specification to introduce hierarchy in the state space, it enables more possibilities for exploiting pattern similarities in the system (section 3),
- Automatic activation of saturation ; the algorithms described in this paper allow the library to enact saturation with minimal user input (sections 4 and 5),
- A *recursive* folding technique that is suitable for very symmetric systems (section 7).

We also show that our openly distributed implementation: *libDDD* [7], is efficient in terms of memory consumption and enables the verification of bigger state spaces.

2 Definitions

We define in this section Data Decision Diagrams (based on [8]) and Set Decision Diagrams (based on [9]).

2.1 Data Decision Diagrams

Data Decision Diagrams (DDD) [8] are a data structure for representing finite sets of assignments sequences of the form $(e_1 := x_1) \cdot (e_2 := x_2) \cdots (e_n := x_n)$ where e_i are variables and x_i are the assigned integer values. When an ordering on the variables is fixed and the values are booleans, DDD coincides with the well-known Binary Decision Diagrams. When the ordering on the variables is the only assumption, DDD correspond closely to Multi-valued Decision Diagrams (MDD)[5].

However DDD assume no variable ordering and, even more, the same variable may occur many times in a same assignment sequence. Moreover, variables are not assumed

to be part of all paths. Therefore, the maximal length of a sequence is not fixed, and sequences of different lengths can coexist in a DDD. This feature is very useful when dealing with dynamic structures like queues.

DDD have two terminals : as usual for decision diagram, 1-leaves stand for accepting terminators and 0-leaves for non-accepting ones. Since there is no assumption on the variable domains, the non-accepted sequences are suppressed from the structure. 0 is considered as the default value and is only used to denote the empty set of sequences. This characteristic of DDD is important as it allows the use of variables of finite domain with *a priori* unknown bounds. In the following, E denotes a set of variables, and for any e in E , $\text{Dom}(e) \subseteq \mathbb{N}$ represents the domain of e .

Definition 1 (Data Decision Diagram). *The set $\mathbb{D}$ of DDD is inductively defined by $d \in \mathbb{D}$ if:*

- $d \in \{0, 1\}$ or
- $d = \langle e, \alpha \rangle$ with:
 - $e \in E$
 - $\alpha : \text{Dom}(e) \rightarrow \mathbb{D}$, such that $\{x \in \text{Dom}(e) \mid \alpha(x) \neq 0\}$ is finite.

We denote $e \xrightarrow{x} d$, the DDD $\langle e, \alpha \rangle$ with $\alpha(x) = d$ and for all $y \neq x$, $\alpha(y) = 0$.

Although no ordering constraints are given, to ensure existence of a canonic representation, DDD only represent sets of *compatible DDD sequences*. Note that the DDD 0 represents the empty set and is therefore compatible with any DDD sequence. The *symmetric compatibility relation* $\approx$ is defined inductively for two DDD sequences:

Definition 2 (Compatible DDD sequences). *We call DDD sequence a DDD of the form $e_1 \xrightarrow{x_1} e_2 \xrightarrow{x_2} \dots 1$. Let s_1, s_2 be two sequences, s_1 is compatible with s_2 , noted $s_1 \approx s_2$ iff.:*

- $s_1 = 1 \wedge s_2 = 1$ or
- $s_1 = e \xrightarrow{x} d \wedge s_2 = e' \xrightarrow{x'} d'$ such that $\begin{cases} e = e' \wedge \\ x = x' \Rightarrow d \approx d' \end{cases}$

As usual, DDD are encoded as (shared) decision trees (see Fig. 1 for an example). Hence, a DDD of the form $\langle e, \alpha \rangle$ is encoded by a node labeled e and for each $x \in \text{Dom}(e)$ such that $\alpha(x) \neq 0$, there is an arc from this node to the root of $\alpha(x)$. By the definition 1, from a node $\langle e, \alpha \rangle$ there can be at most one arc labeled by $x \in \text{Dom}(e)$ and leading to $\alpha(x)$. This may cause conflicts when computing the union of two DDD, if the sequences they contain are incompatible, so care must be taken on the operations performed.

DDD are equipped with the classical set-theoretic operations (union, intersection, set difference). They also offer a concatenation operation $d_1 \cdot d_2$ which replaces 1 terminals of d_1 by d_2 . It corresponds to a cartesian product. In addition, homomorphisms are defined to allow flexibility in the definition of application specific operations.

A basic homomorphism is a mapping Φ from $\mathbb{D}$ to $\mathbb{D}$ such that $\Phi(0) = 0$ and $\Phi(d + d') = \Phi(d) + \Phi(d'), \forall d, d' \in \mathbb{D}$. The sum $+$ and the composition $\circ$ of two homomorphisms are homomorphisms. Some basic homomorphisms are hard-coded. For instance, the

homomorphism $d * Id$ where $d \in \mathbb{D}$, $*$ stands for the intersection and Id for the identity, allows to select the sequences belonging to d : it is a homomorphism that can be applied to any d' yielding $d * Id(d') = d * d'$. The homomorphisms $d * Id$ and $Id * d$ permit to left or right concatenate sequences. We widely use the left concatenation that adds a single assignment ($e := x$), noted $e \xrightarrow{x} Id$.

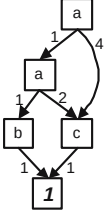


Fig. 1. DDD for
 $a \xrightarrow{1} a \xrightarrow{1} b \xrightarrow{1} 1$
 $+ a \xrightarrow{4} c \xrightarrow{1} 1$
 $+ a \xrightarrow{1} a \xrightarrow{2} c \xrightarrow{1} 1$

We also have a **transitive closure** $*$ unary operator that allows to perform a fixpoint computation. For any homomorphism h , $h^*(d)$, $d \in \mathbb{D}$ is evaluated by repeating $d \leftarrow h(d)$ until a fixpoint is reached. In other words, $h^*(d) = h^n(d)$ where n is the smallest integer such that $h^n(d) = h^{n-1}(d)$. This computation may not terminate (e.g. h increments a variable). However, if it does, then $h^* = h^n$ with n finite. Thus, h^* is itself an inductive homomorphism. This operator is usually applied to $Id + h$ instead of h , allowing to cumulate newly reached paths in the result.

Furthermore, application-specific mappings can be defined by *inductive* homomorphisms. An inductive homomorphism Φ is defined by its evaluation on the 1 terminal $\Phi(1) \in \mathbb{D}$, and its evaluation $\Phi' = \Phi(e, x)$ for any $e \in E$ and any $x \in \text{Dom}(e)$. Φ' is itself a (possibly inductive) homomorphism, that will be applied on the successor node d . The result of $\Phi(\langle e, \alpha \rangle)$ is then defined as $\sum_{(x,d) \in \alpha} \Phi(e, x)(d)$, where

$\sum$ represents a union. We give examples of inductive homomorphisms in section 3 which introduces a simple labeled P/T net formalism.

2.2 Set Decision Diagrams

Set Decision Diagrams (SDD) [9], are shared decision diagrams in which arcs of the structure are labeled by a *set* of values, instead of a single valuation. This set may itself be represented by an SDD or DDD, thus when labels are SDD, we think of them as hierarchical decision diagrams. This section presents the definition of SDD, which has been modified from [9] for more clarity (although it is identical in effects).

Set Decision Diagrams (SDD) are data structures for representing sequences of assignments of the form $e_1 \in a_1; e_2 \in a_2; \dots e_n \in a_n$ where e_i are variables and a_i are sets of values.

SDD can also be seen as a different representation of the DDD defined as:

$\bigcup_{x_1 \in a_1} \dots \bigcup_{x_n \in a_n} e_1 \xrightarrow{x_1} \dots e_n \xrightarrow{x_n} 1$, however since a_i are not required to be finite, SDD are more expressive than DDD.

We assume no variable ordering, and the same variable can occur several times in an assignment sequence. We define the usual terminal 1 to represent accepting sequences. The terminal 0 is also introduced and represents the empty set of assignment sequences. In the following, E denotes a set of variables, and for any e in E , $\text{Dom}(e)$ represents the domain of e which may be infinite.

Definition 3 (Set Decision Diagram). The set $\mathbf{S}$ of SDD is inductively defined by $s \in \mathbf{S}$ if:

- $s \in \{0, 1\}$ or
- $s = \langle e, \pi, \alpha \rangle$ with:

- $e \in E$.
- $\pi = \{a_0, \dots, a_n\}$ is a finite partition of $\text{Dom}(e)$, i.e. $\text{Dom}(e) = a_0 \uplus \dots \uplus a_n$ where $\uplus$ is the disjoint union. We further assume $\forall i, a_i \neq \emptyset$, and n finite.
- $\alpha : \pi \rightarrow \mathbb{S}$, such that $\forall i \neq j, \alpha(a_i) \neq \alpha(a_j)$.

We will simply note $s = \langle e, \alpha \rangle$ the node $\langle e, \pi, \alpha \rangle$ as α implicitly defines π . We denote $e \xrightarrow{a} d$, the SDD (e, α) with $\alpha(a) = d, \alpha(\text{Dom}(e) \setminus a) = 0$. By convention, when it exists, the element of the partition π that maps to the SDD 0 is not represented.

SDD are canonized by construction through the union operator. This definition ensures canonicity of SDD, as π is a partition and that no two arcs from a node may lead to the same SDD. Therefore any value x of $\text{Dom}(e)$ is represented on at most one arc, and any time we are about to construct $e \xrightarrow{a} d + e \xrightarrow{a'} d$, we will construct an arc $e \xrightarrow{a \cup a'} d$ instead. This ensures that any set of assignment sequences has a unique SDD representation.

The finite size of the partition π ensures we can store α as a finite set of pairs (a_i, d_i) , and let π be implicitly defined by α .

Although simple, this definition allows to construct rich and complex data :

- The definition supports domains of infinite size (e.g. $\text{Dom}(e) = \mathbb{R}$), provided that the partition size remains finite (e.g. $[0..3], [3..+\infty]$). This feature could be used to model clocks for instance (as in [10]).
- $\mathbb{S}$ or $\mathbb{D}$ can be used as the domain of variables, introducing hierarchy in the data structure. In the rest of the paper we will focus on this use case, and consider that the SDD variables we manipulate are exclusively of domain $\mathbb{S}$ or $\mathbb{D}$.

Like DDD, to handle paths of variable lengths, SDD are required to represent a set of compatible assignment sequences. An operation over SDD is said partially defined if it may produce incompatible sequences in the result.

Definition 4 (Compatible SDD sequences). An SDD sequence is an SDD of the form $e_0 \xrightarrow{a_0} \dots \xrightarrow{a_n} 1$. Let s_1, s_2 be two sequences, $s_1 \approx s_2$ iff.:

- $s_1 = 1 \wedge s_2 = 1$
- $s_1 = e \xrightarrow{a} d \wedge s_2 = e' \xrightarrow{a'} d'$ such that
$$\begin{cases} e = e' \\ \wedge a \approx a' \\ \wedge a \cap a' \neq \emptyset \Rightarrow d \approx d' \end{cases}$$

Compatibility is a symmetric property. The $a \approx a'$ is defined as SDD compatibility if $a, a' \in \mathbb{S}$ or DDD compatibility if $a, a' \in \mathbb{D}$. DDD and SDD are incompatible. Other possible referenced types should define their own notion of compatibility.

2.3 SDD Operations

SDD support standard set theoretic operations (union, intersection, set difference) for compatible SDD. Like DDD they also support concatenation, as well as a variant of inductive homomorphisms. Some built-in basic homomorphisms (e.g. $d * Id$) are also provided similarly to DDD.

To define a family of inductive homomorphisms Φ , one has just to set the homomorphisms for the symbolic expression $\Phi(e, x)$ for any variable e and set $x \subseteq \text{Dom}(e)$ and the SDD $\Phi(1)$. The application of an inductive homomorphism Φ to a node $s = \langle e, \alpha \rangle$ is then obtained by $\Phi(s) = \sum_{(x,d) \in \alpha} \Phi(e, x)(d)$.

It should be noted that this definition differs from the DDD inductive homomorphism in that $\Phi(e, x)$ is defined over the sets $x \subseteq \text{Dom}(e)$. This is a fundamental difference as it requires Φ to be defined in an ensemblist way: we cannot define the evaluation of Φ over a single value of e . However Φ must be defined for the set containing any single value. If the user only defined $\Phi(e, x)$ with $x \in \text{Dom}(e)$, since the a_i may be infinite, evaluation could be impossible. Even when $\text{Dom}(e) = \mathbb{ID}$, a element-wise definition would force to use an explicit evaluation mechanism, which is not viable when a_i is large (e.g. $|a_i| > 10^7$).

Furthermore, let Φ_1, Φ_2 be two homomorphisms. Then $\Phi_1 + \Phi_2$, $\Phi_1 \circ \Phi_2$ and Φ_1^* (transitive closure) are homomorphisms.

We also now define a *local* construction, as an inductive homomorphism. Let $var \in E$ designate a target variable, and h be a SDD or DDD homomorphism (depending on $\text{Dom}(var)$) that can be applied to any $x \subseteq \text{Dom}(var)$,

$$\begin{aligned} local(h, var)(e, x) = \\ \begin{cases} e \xrightarrow{h(x)} Id & \text{if } e = var \\ e \xrightarrow{x} local(h, var) & \text{otherwise} \end{cases} \\ local(h, var)(1) = 0 \end{aligned}$$

This construction is built-in, and gives a lot of structural information on the operation. As we will see in section 5, specific rewriting rules will allow to optimize evaluation of *local* constructions.

3 Model Checking with Set Decision Diagrams

To build a model checker for a given formalism using SDD, one needs to perform the following steps:

1. Define the formalism,
2. Define a representation of states,
3. Define a transition relation using homomorphisms,
4. Define a verification goal.

We exhibit these steps in this section using a simple formalism, labeled P/T nets. Most of what is presented here is valid for other LTS.

1. Defining the Formalism. A unitary *Labeled P/T-Net* is a tuple $\langle P, T, Pre, Post, L, label, m_0 \rangle$ where

- P is a finite set of places,
- T is a finite set of transitions (with $P \cap T = \emptyset$),
- Pre and $Post : P \times T \rightarrow \mathbb{IN}$ are the pre and post functions labeling the arcs.

- L is a set of labels
- $label : T \rightarrow 2^L$ is a function labeling the transitions
- $m_0 \in \mathbb{N}^P$ is the initial marking of the net.

For a transition t , $\bullet t$ (resp. $t\bullet$) denotes the set of places $\{p \in P \mid Pre(p, t) \neq 0\}$ (resp. $\{p \in P \mid Post(p, t) \neq 0\}$). A marking m is an element of $\mathbb{N}^P$. A transition t is enabled in a marking m if for each place p , the condition $Pre(p, t) \leq m(p)$ holds. The firing of an enabled transition t from a marking m leads to a new marking m' defined by $\forall p \in P, m'(p) = m(p) - Pre(p, t) + Post(p, t)$.

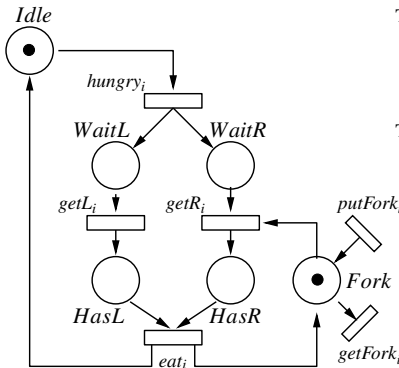
Labeled P/T nets may be composed by synchronization on the transitions that bear the same label. This is a parallel composition noted $\parallel$, with event-based synchronizations that can be interpreted as yielding a new (composite) labeled P/T net. Let $M = M_0 \parallel \dots \parallel M_n$ be such a composite labeled Petri net. Each label of M common to the composed nets M_i gives rise to a *synchronization transition* t_l of M .

Let $\tau_l = \{t_i \mid t_i \in M_i.T \wedge l \in M_i.label(t_i)\}$ represent *parts of the synchronization*, i.e. the set of transitions that bear this label in the subnets M_i . t_l is enabled iff. $\forall t_i \in \tau_l$, t_i is enabled. The effect firing of t_l is obtained by firing all the parts $t_i \in \tau_l$. In the rest of this paper, we will call *labeled Petri net* a unitary or composite net.

Figure 2 presents an example labeled Petri net, for the classical dining philosophers problem. The composite net $P_0 \parallel P_1$ synchronizes transition $P_0.eat$ with $P_1.putFork$ through label $\mathcal{R}_0$ for instance. This transition corresponds to philosopher P_0 synchronously eating and returning philosopher P_1 his fork.

This is a general compositional framework, adapted to the composition of arbitrary labeled transition systems (LTS).

2. Defining the State Representation. Let us consider a representation of a state space of a unitary P/T net in which we use one DDD variable for each place of the system. The domain of place variables is the set of natural numbers. The initial marking for a single place is represented by: $d_p = p \xrightarrow{m_0(p)} 1$. For a given total order on the places of the net, the DDD representing the initial marking is the concatenation of DDD $d_{p_1} \dots d_{p_n}$.



Transition eat_i is synchronized with $putFork_{i+1 \bmod N}$, thus the labels

$$label(putFork_i) = \{\mathcal{R}_{i+1 \bmod N}\}$$

$$label(eat_i) = \{\mathcal{R}_i\}$$

Transition $getL_i$ is synchronized with $getFork_{i+1 \bmod N}$, thus the labels

$$label(getFork_i) = \{\mathcal{L}_{i+1 \bmod N}\}$$

$$label(getL_i) = \{\mathcal{L}_i\}$$

Fig. 2. Labeled P/T net P_i of i^{th} philosopher in the N dining philosophers problem

For instance, the initial state of a philosopher can be represented by : $Fork \xrightarrow{1} HasR \xrightarrow{0} WaitR \xrightarrow{0} HasL \xrightarrow{0} WaitL \xrightarrow{1} Idle \xrightarrow{1} 1$.

To introduce structure in the representation, we introduce the role of parenthesis in the definition of a composite net. We will thus exploit the fact the model is defined as a composition of (relatively independent) parts in our encoding. If we disregard any parenthesizing of the composition we obtain an equivalent “flat” composite net, however using different parenthesizing(s) yields a more hierarchical vision (nested submodules), that can be accurately represented and exploited in our framework.

Definition 5 (Structured state representation). Let M be a labeled P/T net, we inductively define its initial state representation $r(M)$ by :

- If M is a unitary net, we use the encoding $r(M) = d_{p_1} \cdots d_{p_n}$, with $d_p = p \xrightarrow{m_0(p)} 1$.
- If $M = M_1 \parallel M_2$, $r(M) = r(M_1) \cdot r(M_2)$. Thus the parallel composition of two nets will give rise to the concatenation of their representations.
- If $M = (M_1)$, $r(M) = m_{(M_1)} \xrightarrow{r(M_1)} 1$, where $m_{(M_1)}$ is an SDD variable. Thus parenthesizing an expression gives rise to a new level of hierarchy in the representation.

A state is thus encoded hierarchically in accordance with the parenthesized composition definition. If we disregard parenthesizing, we obtain a flat representation using only DDD. We use in our benchmark set many models taken from literature that are defined using “modules”, that is a net $N = (M_1) \parallel \cdots \parallel (M_n)$ where each M_i is a unitary net called a module (yielding a single level of hierarchy in the SDD). Figure 3 shows an example of this type of encoding, where figure 3(a) is an SDD representing the full composite net, and labels of the SDD arcs refer to DDD nodes of figure 3(b).

3. Defining the Transition encoding. The symbolic transition relation is defined arc by arc in a modular way well-adapted to the further combination of arcs of different

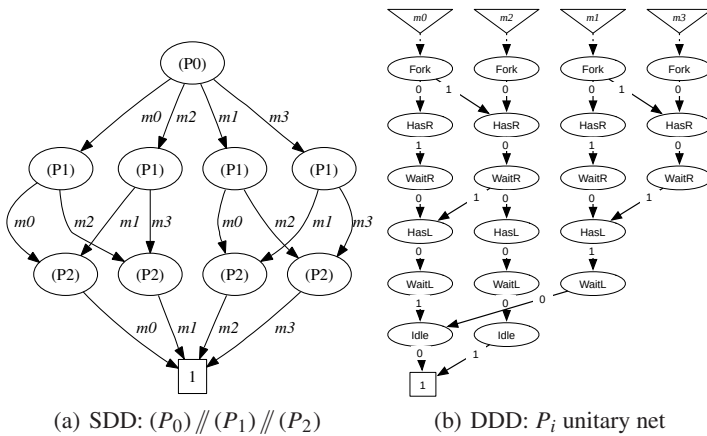


Fig. 3. Hierarchical encoding of the full state-space for 3 philosophers

net sub-classes (e.g. inhibitor arcs, reset arcs, capacity places, queues...). Homomorphisms allowing to represent these extensions were previously defined in [8], and are not presented here for sake of simplicity. The two following homomorphisms are defined to deal respectively with the pre (noted h^-) and post (noted h^+) conditions. Both are parameterized by the connected place (p) as well as the valuation (v) labeling the arc entering or outing p .

$$\left. \begin{array}{l} h^-(p, v)(e, x) = \\ \left\{ \begin{array}{ll} e \xrightarrow{x-v} Id & \text{if } e = p \wedge x \geq v \\ 0 & \text{if } e = p \wedge x < v \\ e \xrightarrow{x} h^-(p, v) & \text{otherwise} \end{array} \right. \\ h^-(p, v)(1) = 0 \end{array} \right| \begin{array}{l} h^+(p, v)(e, x) = \\ \left\{ \begin{array}{ll} e \xrightarrow{x+v} Id & \text{if } e = p \\ e \xrightarrow{x} h^+(p, v) & \text{otherwise} \end{array} \right. \\ h^+(p, v)(1) = 0 \end{array}$$

These basic homomorphisms are composed to form a transition relation. We use $\circ_{h \in H}$ to denote the composition by $\circ$ of the elements h in the set H .

Definition 6 (Inductive homomorphism transition representation). *Let t be a transition of labeled P/T net M . We inductively define its representation as a homomorphisms $h_{Trans}(t)$ by :*

- If M is a unitary net, we use the encoding

$$h_{Trans}(t) = \circ_{p \in t} h^+(p, Post(p, t)) \circ \circ_{p \in t} h^-(p, Pre(p, t))$$

- If $M = (M_1) \parallel \dots \parallel (M_n)$, and t represents a synchronization of transitions on a label $l \in L$. The homomorphism representing t is written :

$$h_{Trans}(t) = \circ_{t_i \in \tau_l} local(h_{Trans}(t_i), m_{(M_i)})$$

For instance the transition $hungry_i$ in the model of Fig. 2, would have as homomorphism : $h_{Trans}(hungry) = h^+(WaitL, 1) \circ h^+(WaitR, 1) \circ h^-(Idle, 1)$. When on a path a precondition is unsatisfied, the h^- homomorphism will return 0, pruning the path from the structure. Thus the h^+ are only applied on the paths such that all preconditions are satisfied.

To handle synchronization of transitions bearing the same label in different nets of a compositional net definition we use the local application construction of SDD homomorphisms. The fact that this definition as a composition of local actions is possible stems from the simple nature of the synchronization schema considered. A transition relation that is decomposable under this form has been called Kronecker-consistent in various papers on MDD by Ciardo et al like [5].

For instance, let us consider the dining philosophers example for $N = 3$, $M = (P_0) \parallel (P_1) \parallel (P_2)$. The transition t_{R_0} is written :

$$\begin{aligned} h_{Trans}(t_{R_0}) &= local(h_{Trans}(eat), m_{(P_0)}) \\ &\quad \circ local(h_{Trans}(putFork), m_{(P_1)}) \\ &= local(h^+(Idle, 1) \circ h^+(Fork, 1) \circ h^-(HasL, 1) \circ h^-(HasR, 1), m_{(P_0)}) \\ &\quad \circ local(h^+(Fork, 1), m_{(P_1)}) \end{aligned}$$

4. Defining the Verification Goal. The last task remaining is to define a set of target (usually undesired) states, and check whether they are reachable, which involves generating the set of reachable states using a *fixpoint* over the transition relation. The user is then free to define a selection inductive homomorphism that only keeps states that verify an atomic property. This is quite simple, using homomorphisms similar to the pre condition (h^-) that do not modify the states they are applied to. Any boolean combination of atomic properties is easily expressed using union, intersection and set difference.

A more complex CTL logic model-checker can then be constructed using nested *fixpoint constructions* over the transition relation or its reverse [2]. Algorithms to produce witness (counter-example) traces also exist [11] and can be implemented using SDD.

4 Transitive Closure : State of the Art

The previous section has allowed us to obtain an encoding of states using SDD and of transitions using homomorphisms. We have concluded with the importance of having an efficient algorithm to obtain the transitive closure or fixpoint of the transition relation over a set of (initial) states, as this procedure is central to the model-checking problem.

Such a transitive closure can be obtained using various algorithms, some of which are presented in algorithm 1. Variant *a* is a naive algorithm, *b* [2] and *c* [4] are algorithms from the literature. Variant *d*, together with automatic optimizations, is our contribution and will be presented in the next section.

Symbolic transitive closure ('91)[2]. Variation *a* is adapted from the natural way of writing a fixpoint with explicit data structures: it uses a set *todo* exclusively containing unexplored states. Notice the slight notation abuse: we note $T(todo)$ when we should note $(\sum_{t \in T} t)(todo)$.

Variant *b* instead applies the transition relation to the full set of currently reached states. Variant *b* is actually much more efficient than variant *a* in practice. This is due to the fact that the size of DD is not directly linked to the number of states encoded, thus the *todo* of variant *a* may actually be much larger in memory. Variant *a* also requires more computations (to get the difference) which are of limited use to produce the final result. Finally, applying the transition relation to states that have been already explored in *b* may actually not be very costly due to the existence of a cache.

Variant *b* is similar to the original way of writing a fixpoint as found in [2]. Note that the standard encoding of a transition relation uses a DD with two DD variables (before and after the transition) for each DD variable of the state. Keeping each transition DD isolated induces a high time overhead, as different transitions then cannot share traversal. Thus the union of transitions T is stored as a DD, in other approaches than our DDD/SDD. However, simply computing this union T has been shown in some cases to be intractable.

Chaining ('95)[4]. An intermediate approach is to use clusters. Transition clusters are defined and a DD representing each cluster is computed using union. This produces smaller DD, that represent the transition relation in parts. The transitive closure is then

Algorithm 1. Four variants of a transitive closure loop**Data:** $\{Hom\} T$: the set of transitions encoded as h_{Trans} homomorphisms**S** m_0 : initial state encoded as $r(M)$ SDD**S** $todo$: new states to explore**S** $reach$: reachable states

a) Explicit reachability style

```

begin
   $todo := m_0$ 
   $reach := m_0$ 
  while  $todo \neq 0$  do
    S  $tmp := T(todo)$ 
     $todo := tmp \setminus reach$ 
     $reach := reach + tmp$ 
  end

```

b) Standard symbolic BFS loop

```

begin
   $todo := m_0$ 
   $reach := 0$ 
  while  $todo \neq reach$  do
     $reach := todo$ 
     $todo := todo + T(todo) \equiv (T + Id)(todo)$ 
  end

```

c) Chaining loop

```

begin
   $todo := m_0$ 
   $reach := 0$ 
  while  $todo \neq reach$  do
     $reach := todo$ 
    for  $t \in T$  do
       $todo := (t + Id)(todo)$ 
    end
  end

```

d) Saturation enabled

```

begin
   $reach := (T + Id)^*(m_0)$ 
end

```

obtained by algorithm *c*, where each t represents a cluster. Note that this algorithm no longer explores states in a strict BFS order, as when t_2 is applied after t_1 , it may discover successors of states obtained by the application of t_1 . The clusters are defined in [4] using structural heuristics that rely on the Petri net definition of the model, and try to maximize independence of clusters. This may allow to converge faster than in *a* or *b* which will need as many iterations as the state-space is deep. While this variant relies on a heuristic, it has empirically been shown to be much better than *b*.

Saturation (*01)[5]. Finally the saturation method is empirically an order of magnitude better than *c*. Saturation consists in constructing clusters based on the highest DD variable that is used by a transition. Any time a DD node of the state space representation is modified by a transition it is (re)saturated, that is the cluster that corresponds to this variable is applied to the node until a fixpoint is reached. When saturating a node, if lower nodes in the data structure are modified they will themselves be (re)saturated. This recursive algorithm can be seen as particular application order of the transition clusters that is adapted to the DD representation of state space, instead of exploring in BFS order the states.

The saturation algorithm is not represented in the algorithm variants figure because it is described (in [5]) on a full page that defines complex mutually recursive procedures, and would not fit here. Furthermore, DD packages such as *cudd* or *Buddy* [12,13] do not provide in their public API the possibility of such fine manipulation of the evaluation procedure, so the algorithm of [5] cannot be easily implemented using those packages.

Our Contribution. All these algorithm variants, including saturation (see [9]), can be implemented using SDD. However we introduce in this paper a more natural way of expressing a fixpoint through the h^* unary operator, presented in variant d . The application order of transitions is not specified by the user in this version, leaving it up to the library to decide how to best compute the result. By default, the library will thus apply the most efficient algorithm currently available: saturation. We thus overcome the limits of other DD packages, by implementing saturation *inside* the library.

5 Automating Saturation

This section presents how using simple rewriting rules we automatically create a saturation effect. This allows to embed the complex logic of this algorithm in the library, offering the power of this technique at no additional cost to users. At the heart of this optimization is the property of *local invariance*.

5.1 Local Invariance

A minimal structural information is needed for saturation to be possible: the highest variable operations need to be applied to must be known. To this end we define :

Definition 7 (Locally invariant homomorphism). *An homomorphism h is locally invariant on variable e iff*

$$\forall s = \langle e, \alpha \rangle \in \mathbb{D} \cup \mathbb{S}, h(s) = \sum_{(x,d) \in \alpha} e \xrightarrow{x} h(d)$$

Concretely, this means that the application of h doesn't modify the structure of nodes of variable e , and h is not modified by traversing these nodes. The variable e is a “don't care” w.r.t. operation h , it is neither written nor read by h . A standard DD encoding [5] of h applied to this variable would produce the identity. The identity homomorphism Id is locally invariant on all variables.

For an inductive homomorphism h locally invariant on e , it means that $h(e, x) = e \xrightarrow{x} h$. A user defining an inductive homomorphism h should provide a predicate $Skip(e)$ that returns *true* if h is locally invariant on variable e . This minimal information will be used to reorder the application of homomorphisms to produce a saturation effect. It is not difficult when writing a homomorphism to define this $Skip$ predicate since the useful variables are known, it actually reduces the number of tests that need to be written.

For example, the h^+ and h^- homomorphisms of section 3 can exhibit the locality of their effect on the state signature by defining $Skip$, which removes the test $e = p$ w.r.t. the previous definition since p is the only variable that is not *skipped*:

$$\left. \begin{array}{l} h^-(p, v)(e, x) = \\ \left\{ \begin{array}{ll} e \xrightarrow{x-v} Id & \text{if } x \geq v \\ 0 & \text{if } x < v \end{array} \right. \\ h^-.Skip(e) = (e \neq p) \\ h^-(p, v)(1) = 0 \end{array} \right| \begin{array}{l} h^+(p, v)(e, x) = e \xrightarrow{x+v} Id \\ h^+.Skip(e) = (e \neq p) \\ h^+(p, v)(1) = 0 \end{array}$$

An inductive homomorphism Φ 's application to $s = \langle e, \alpha \rangle$ is defined by $\Phi(s) = \sum_{(x,d) \in \alpha} \Phi(e, x)(d)$. But when Φ is invariant on e , computation of this union produces the expression $\sum_{(x,d) \in \alpha} e \xrightarrow{x} \Phi(d)$. This result is known beforehand thanks to the predicate *Skip*.

From an implementation point of view this allows us to create a new node directly by copying the structure of the original node and modifying it in place. Indeed the application of Φ will at worst remove some arcs. If a $\Phi(d)$ produces the 0 terminal, we prune the arc. Else, if two $\Phi(d)$ applications return the same value in SDD setting, we need to fuse the arcs into an arc labeled by the union of the arc values. We thus avoid computing the expression $\sum_{(x,d) \in \alpha} \Phi(e, x)(d)$, which involves creation of intermediate single arc nodes $e \xrightarrow{x} \dots$ and their subsequent union. The impact on performances of this "in place" evaluation is already measurable, but more importantly it enables the next step of rewriting rules.

5.2 Union and Composition

For built-in homomorphisms the value of the *Skip* predicate can be computed by querying their operands: homomorphisms constructed using union, composition and fixpoint of other homomorphisms, are locally invariant on variable e if their operands are themselves invariant on e .

This property derives from the definition (given in [8,9]) of the basic set theory operations on DDD and SDD. Indeed for two homomorphisms h and h' locally invariant on variable e we have: $\forall s = \langle e, \alpha \rangle \in \mathbb{D} \cup \mathbb{S}$,

$$\begin{aligned} (h + h')(s) &= h(s) + h'(s) \\ &= \sum_{(x,d) \in \alpha} e \xrightarrow{x} h(d) + \sum_{(x,d) \in \alpha} e \xrightarrow{x} h'(d) \\ &= \sum_{(x,d) \in \alpha} e \xrightarrow{x} h(d) + h'(d) \\ &= \sum_{(x,d) \in \alpha} e \xrightarrow{x} (h + h')(d) \end{aligned}$$

A similar reasoning can be used to prove the property for composition.

It allows homomorphisms nested in a union to share traversal of the nodes at the top of the structure as long as they are locally invariant. When they no longer *Skip* variables, the usual evaluation definition $h(s) + h'(s)$ is used to affect the current node. Until then, the shared traversal implies better time complexity and better memory complexity as they also share cache entries.

We further support natively the n-ary union of homomorphisms. This allows to dynamically create clusters by top application level as the union evaluation travels downwards on nodes. When evaluating an n-ary union $H(s) = \sum_i h_i(s)$ on a node $s = \langle e, \alpha \rangle$ we partition its operands into $F = \{h_i | h_i.Skip(e)\}$ and $G = \{h_i | \neg h_i.Skip(e)\}$. We then rewrite the union $H(s) = (\sum_{h \in F} h)(s) + (\sum_{h \in G} h)(s)$, or more simply $H(s) = F(s) + G(s)$. The F union is thus locally invariant on e and will continue evaluation as a block. The G part is evaluated using the standard definition $G(s) = \sum_{h \in G} h(s)$.

Thus the minimal *Skip* predicate allows to automatically create clusters of operations by adapting to the structure of the SDD it is applied to. We still have no requirements

on the order of variables, as the clusters can be created dynamically. To obtain efficiency, the partitions $F + G$ are cached, as the structure of the SDD typically has limited variation during construction. Thus the partitions for an nary union are computed at most once per variable instead of once per node.

The computation using the definition of $H(s) = \sum_i h_i(s)$ requires each h_i to separately traverse s , and forces to fully rebuild all the $h_i(s)$. In contrast, applying a union H allows sharing of traversals of the SDD for its elements, as operations are carried to their application level in clusters before being applied. Thus, when a strict BFS progression (like algorithm 1.b) is required this new evaluation mechanism has a significant effect on performance.

5.3 Fixpoint

With the rewriting rule of a union $H = F + G$ we have defined, we can now examine the rewriting of an expression $(H + Id)^*(d)$ as found in algorithm 1.d :

$$\begin{aligned} (H + Id)^*(s) &= (F + G + Id)^*(s) \\ &= (G + Id + (F + Id)^*)^*(s) \end{aligned}$$

The $(F + Id)^*$ block by definition is locally invariant on the current variable. Thus it is directly propagated to the successor nodes, where it will recursively be evaluated using the same definition as $(H + Id)^*$.

The remaining fixpoint over G homomorphisms can be evaluated using the chaining operation order (see algorithm 1.c), which is reported empirically more effective than other approaches [14], a result also confirmed in our experiments.

The chaining application order algorithm 1.c can be written compactly in SDD as :

$$reach = (\bigcirc_{t \in T} (t + Id))^*(s_0)$$

We thus finally rewrite:

$$(H + Id)^*(s) = (\bigcirc_{g \in G} (g + Id) \circ (F + Id)^*)^*(s)$$

5.4 Local Applications

We have additional rewriting rules specific to SDD homomorphisms and the *local* construction (see section 2.3) :

$$\begin{aligned} local(h, var)(e, x) &= e \xrightarrow{h(x)} Id \\ local(h, var).Skip(e) &= (r \neq var) \\ local(h, var)(1) &= 0 \end{aligned}$$

Note that h is a homomorphism, and its application is thus linear to the values in x . Further a local operation can only affect a single level of the structure (defined by var). We can thus define the following rewriting rules, exploiting the locality of the operation:

- (1) $local(h, v) \circ local(h', v) = local(h \circ h', v)$
- (2) $local(h, v) + local(h', v) = local(h + h', v)$

- (3) $v \neq v' \implies local(h, v) \circ local(h', v') = local(h', v') \circ local(h, v)$
 (4) $(local(h, v) + Id)^* = local((h + Id)^*, v)$

Expressions (1) and (2) come from the fact that a local operation is locally invariant on all variables except v . Expression (3) asserts commutativity of composition of local operations, when they do not concern the same variable. Indeed, the effect of applying $local(h, v)$ is only to modify the state of variable v , so modifying v then v' or modifying v' then v has the same overall effect. Thus two local applications that do not concern the same variable are independent. We exploit this rewriting rule when considering a composition of $local$ to maximize applications of the rule (1), by sorting the composition by application variable. A final rewriting rule (4) is used to allow nested propagation of the fixpoint. It derives directly from rules (1) and (2).

With these additional rewriting rules defined, we slightly change the rewriting of $(H + Id)^*(s)$ for node $s = \langle e, \alpha \rangle$: we consider $H(s) = F(s) + L(s) + G(s)$ where F contains the locally invariant part, $L = local(l, e)$ represents the operations purely local to the current variable e (if any), and G contains operations which affect the value of e (and possibly also other variables below). Thanks to rule (4) above, we can write :

$$\begin{aligned}
 (H + Id)^*(s) &= (F + L + G + Id)^*(s) \\
 &= (G + Id + (L + Id)^* + (F + Id)^*)^*(s) \\
 &= (\bigcirc_{g \in G} (g + Id) \circ local((l + Id)^*, e) \circ (F + Id)^*)^*(s)
 \end{aligned}$$

As the next section presenting performance evaluations will show, this saturation style application order heuristically allows to gain an order of magnitude in the size of models that can be treated.

6 Performances of Automatic Saturation

Impact of Propagation. We have first measured how the propagation alone impacts on memory size, that is without automatic saturation. We have thus measured the memory footprint when using a chaining loop with propagation enabled or not. We have observed a gain from 15% to 50%, with an average of about 40%. This is due to the shared traversal of homomorphisms when they are propagated, thus inducing much less creation of intermediary nodes.

Impact of Hierarchy and Automatic Saturation. Table 1 shows the results obtained (on a Xeon @ 1.83GHz with 4GB of memory) when generating the state spaces of several models with automatic saturation (Algo. 1.d) compared to those obtained using a standard chaining loop (Algo. 1.c). Moreover, we measured how hierarchical encoding of state spaces perform compared to flat encoding (DDD).

We have run the benchmarks on 4 parametrized models, with different sizes: the well-known Dining Philosophers and Kanban models; a model of the slotted ring protocol; a model of a flexible manufacturing system. We have also benchmarked a LOTOS specification obtained from a true industrial case-study (it was generated automatically from a LOTOS specification – 8,500 lines of LOTOS code + 3,000 lines of C code – by Hubert Garavel from INRIA).

Table 1. Impact of hierarchical decision diagrams and automatic saturation

		Final #		Hierarchical Chaining Loop			Flat Automatic Sat.			Hierarchical Automatic Sat.		
Model Size	States #	DDD	SDD	T. (s)	Mem. (MB)	Peak #	T. (s)	Mem. (MB)	Peak #	T. (s)	Mem. (MB)	Peak #
LOTOS Specification												
	9.8e+21	–	1085	–	–	–	–	–	–	1.47	74.0	110e+3
Dining Philosophers												
100	4.9e+62	2792	419	1.9	112	276e+3	0.2	20	18040	0.07	5.2	4614
200	2.5e+125	5589	819	7.9	446	1.1e+6	0.7	58.1	36241	0.2	10.6	9216
1000	9.2e+626	27989	4019	–	–	–	14	1108	182e+3	4.3	115	46015
4000	7e+2507	–	16019	–	–	–	–	–	–	77	1488	184e+3
Slotted Ring Protocol												
10	8.3e+09	1283	105	1.1	48	90043	0.2	16	31501	0.03	3.5	3743
50	1.7e+52	29403	1345	–	–	–	22	1054	2.4e+6	5.1	209	461e+3
100	2.6e+105	–	5145	–	–	–	–	–	–	22	816	1.7e+6
150	4.5e+158	–	11445	–	–	–	–	–	–	60	2466	5.6e+6
Kanban												
100	1.7e+19	11419	511	12	145	264e+3	2.9	132	309e+3	0.4	11	14817
200	3.2e+22	42819	1011	96	563	1e+6	19	809	1.9e+6	2.2	37	46617
300	2.6e+24	94219	1511	–	–	–	60	2482	5.7e+6	7	78	104e+3
700	2.8+28	–	3511	–	–	–	–	–	–	95	397	523e+3
Flexible Manufacturing System												
50	4.2e+17	8822	917	13	430	530e+3	2.7	105	222e+3	0.4	16	23287
100	2.7e+21	32622	1817	–	–	–	19	627	1.3e+6	1.9	50	76587
150	4.8e+23	71422	2717	–	–	–	62	1875	3.8e+6	5.3	105	160e+3
300	3.6e+27	–	5417	–	–	–	–	–	–	33	386	590e+3

All “—” entries indicate that the state space’s generation did not finish because of the exhaustion of the computer’s main memory.

The “*Final*” grey columns show the final number of decision diagram nodes needed to encode the state spaces for hierarchical (SDD) and flat (DDD) encoding. Clearly, flat DD need an order of magnitude of more nodes to store a state space. This shows how well hierarchy factorizes state spaces. The good performances of hierarchy also show that using a structured specification can help detect similarity of behavior in parts of a model, enabling sharing of their state space representation (see figure 3).

But the gains from enabling saturation are even more important than the gains from using hierarchy on this example set. Indeed, saturation allows to mostly overcome the “peak effect” problem. Thus “*Flat Automatic Saturation*” performs better (in both time and memory) than “*Hierarchical Chaining Loop*”.

¹ We haven’t reported results for flat DDs with a chaining loop generation algorithm as they were nearly always unable to handle models of big size.

As expected, mixing hierarchical encoding and saturation brings the best results: this combination enables the generation of much larger models than other methods on a smaller memory footprint and in less time.

7 Recursive Folding

In this section we show how SDD allow in some cases to gain an order of complexity: we define a solution to the state-space generation of the philosophers problem which has complexity in time and memory *logarithmic* to the number of philosophers. The philosophers system is highly symmetric, and is thus well-adapted to techniques that exploit this symmetry. We show how SDD allow to capture this symmetry by an adapted hierarchical encoding of the state-space. The crucial idea is to use a recursive folding of the model with n levels of depth for 2^n philosophers ².

7.1 Initial State

Instead of $(P_0) \parallel (P_1) \parallel (P_2) \parallel (P_3)$ which is the parenthesizing that is assumed by default, we parenthesize our composition $((P_0) \parallel (P_1)) \parallel ((P_2) \parallel (P_3))$. We will thus introduce $n + 2$ levels of hierarchy to represent 2^n philosophers, each level corresponding to a parenthesis group. Since each parenthesis group $((X) \parallel (Y))$ only contains one composition $\parallel$, its SDD will contain two variables that correspond to the states of (X) and (Y) .

The innermost level (level 0, corresponding to the most nested parenthesis of the composition) contains a variable of domain the states of a single philosopher. The most external parenthesis group will be used to close the loop, i.e. connect the first and last philosophers. Hence level 0 represents a single philosopher, level 1 represents the states of two philosophers, and level i represents the states of 2^i philosophers.

The magic in this representation is that each half of the philosophers at any level behaves in the same way as the other half : it's really $((P_0) \parallel (P_0)) \parallel ((P_0) \parallel (P_0))$. Thus sharing is extremely high : the initial state of the system for 2^n philosophers only requires $2n + k$ ($k \in \mathbb{N}$) nodes to be represented.

Let $P_0 = Fork \xrightarrow{1} HasR \xrightarrow{0} WaitR \xrightarrow{0} HasL \xrightarrow{0} WaitL \xrightarrow{0} Idle \xrightarrow{1} 1$ represent the states of a single philosopher as a DDD (as in section 3). Let M_k represent the states of 2^k philosophers using the recursive parenthesizing scheme. Following our definitions of the previous section, M_k is defined inductively by :

$$M_k = h_0 \xrightarrow{M_{k-1}} h_1 \xrightarrow{M_{k-1}} 1 \quad M_0 = p \xrightarrow{P_0} 1$$

The most external parenthesis group yields a last variable noted $h_{(M_n)}$ such that $r((M_n)) = h_{(M_n)} \xrightarrow{M_n} 1$. We have thus defined 4 variables: $h_{(M_n)}$ for the external parenthesis, h_0 and h_1 for intermediate levels, and p for the last level ($\text{Dom}(p) \subseteq \mathbb{ID}$).

7.2 Transition Relation

We define the SDD homomorphisms f and l to work respectively on the first and last philosopher modules of a submodule, as they communicate by a synchronization transition.

² We thank Jean-Michel Couvreur for fruitful input on this idea.

$$\begin{array}{c|c}
\begin{array}{l}
f(h)(e, x) = \\
\left\{ \begin{array}{ll} e \xrightarrow{h(x)} Id & \text{if } e = p \\ e \xrightarrow{f(h)(x)} Id & \text{if } e = h_0 \end{array} \right. \\
f.Skip(e) = (e \neq p) \wedge (e \neq h_0) \\
f(h)(1) = 0
\end{array} &
\begin{array}{l}
l(h)(e, x) = \\
\left\{ \begin{array}{ll} e \xrightarrow{h(x)} Id & \text{if } e = p \\ e \xrightarrow{l(h)(x)} Id & \text{if } e = h_1 \end{array} \right. \\
l.Skip(e) = (e \neq p) \wedge (e \neq h_1) \\
l(h)(1) = 0
\end{array}
\end{array}$$

We then need to take into account that all modules have the same transitions. Transitions that are purely local to a philosopher module are unioned and stored in a homomorphism which will be noted $\mathcal{L}$ (in fact only *hungry* is purely local). We note $\Pi_i(s)$ the part of a synchronization transition s_L created for label L that concerns the *current* philosopher module P_i and $\Pi_{i+1}(s)$ the part of s_L that concerns $P_{i+1 \bmod N}$ the right hand neighbor of P_i . We note S the set of synchronization transitions, induced by the labels $\mathcal{L}_i$ and $\mathcal{R}_i$.

$$\text{Let } \tau_{loop} = Id + \sum_{s \in S} l(\Pi_i(s)) \circ f(\Pi_{i+1}(s))$$

τ_{loop} is an SDD homomorphism operation defined to “close the loop”, that materializes that the last philosophers right hand neighbor is the first philosopher. Our main firing operation that controls the saturation is τ defined as follows :

$$\begin{array}{ll}
\tau(e, x) = & \\
\left\{ \begin{array}{ll} e \xrightarrow{(\tau \circ \tau_{loop})^*(x)} Id & \text{if } e = h(M_n) \\ e \xrightarrow{\tau^*(x)} \tau + \sum_{s \in S} e \xrightarrow{\tau^* \circ l(\Pi_i(s))} \tau \circ f(\Pi_{i+1}(s)) & \text{if } e = h_0 \\ e \xrightarrow{\tau^*(x)} Id & \text{if } e = h_1 \\ e \xrightarrow{\mathcal{L}^*(x)} Id & \text{if } e = p \end{array} \right. \\
\tau(t)(1) = 0 &
\end{array}$$

We can easily adapt this encoding to treat an arbitrary number n of philosophers instead of powers of 2, by decomposing n into it's binary encoding. For instance, for $5 = 2^0 + 2^2$ philosophers $((P0) \parallel ((P1 \parallel P2) \parallel (P3 \parallel P4)))$ Such unbalanced depth in the data structure is gracefully handled by the homogeneity of our operation definitions, and does not increase computational complexity.

7.3 Experimentation

We show in table 2 how SDD provide an elegant solution to the state-space generation of the philosophers problem, for up to 2^{20000} philosophers. The complexity both in time and space is roughly linear to n , with empirically $8n$ nodes and $12n$ arcs required to represent the final state-space of 2^n philos.

The solution presented here is specific to the philosophers problem, though it can be adapted to other symmetric problems. Its efficiency here is essentially due to the inherent properties of the model under study. In particular the strong locality, symmetry and the fact that even in a BDD/DDD representation, adding philosophers does not increase the “width” of the DDD representation – only it's height –, are the key factors.

Table 2. Performances of recursive folding with 2^n philosophers . The states count is noted *N/A* when the large number library GNU Multiple Precision (GMP) we use reports an overflow.

Nb. Philosophers	States	Time (s)	Final		Peak	
			SDD	DDD	SDD	DDD
2^{10}	1.02337e+642	0.0	83	31	717	97
2^{31}	1.63233e+1346392620	0.02	251	31	2250	97
2^{1000}	<i>N/A</i>	0.81	8003	31	72987	97
2^{10000}	<i>N/A</i>	9.85	80003	31	729987	97
2^{20000}	<i>N/A</i>	20.61	160003	31	1459987	97

The difficulty in generalizing the results of this example, is that we exploit in the definition of the transition relation the fact that all philosophers have the same behavior, and the circular way they are synchronized. In other words, our formalism is not well adapted to scaling to 2^n , because it lacks an inductive definition of the problem that we could capture automatically. While a simple use of the parenthesizing scheme described in section 3 would produce overall the same effects, the recursive homogeneity captured by τ would be lost. We would then have linear complexity w.r.t. to the number of philosophers, when computing our rewriting rules, which is not viable to scale up to 2^{20000} as we no longer can have overall logarithmic complexity.

Thus our current research direction consists in defining a formalism (e.g. a particular family of Petri nets) such that we could recognize this pattern and obtain the recursive encoding naturally.

However, this example reveals that SDD are potentially exponentially more powerful than other decision diagram variants.

8 Conclusion

In this paper, we have presented the latest evolutions of hierarchical Set Decision Diagrams (SDD), that are suitable to master the complexity of very large systems. We think that such diagrams are well-adapted to process hierarchical high-level specifications such as Net-within-Nets [15] or CO-OPN [16].

We have presented how we optimize evaluation of user homomorphisms to automatically producing a saturation effect. Moreover, this automation is done at a low cost for users since it uses a *Skip* predicate that is easy to define. We thus generalize extremely efficient saturation approach of Ciardo et al. [5] by giving a definition that is entirely based on the structure of the decision diagram and the operations encoded, instead of involving a given formalism. Furthermore, the automatic activation of saturation allows users to concentrate on defining the state and transition encoding.

Also, we have shown how recursive folding allows in very efficient and elegant manner to generate state spaces of regular and symmetric models, with up to 2^{20000} philosophers in our example. Although generalization of this application example is left to further research, it exhibits the potentially exponentially better encoding SDD provide over other DD variants for regular examples.

SDD and the optimizations described are implemented in `libddd`, a C++ library freely available under the terms of GNU LGPL. With growing maturity since the initial prototype developed in 2001 and described in [8], `libddd` is today a viable alternative to Buddy [13] or CUDD [12] for developers wishing to take advantage of symbolic encodings to build a model-checker.

References

1. Bryant, R.: Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8), 677–691 (1986)
2. Burch, J., Clarke, E., McMillan, K.: Symbolic model checking: 10^{20} states and beyond. (Special issue for best papers from LICS90) *Information and Computation* 98(2), 153–181 (1992)
3. Bollig, B., Wegener, I.: Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Trans. Comput.* 45(9), 993–1002 (1996)
4. Roig, O., Cortadella, J., Pastor, E.: Verification of asynchronous circuits by BDD-based model checking of Petri nets. In: DeMichelis, G., Díaz, M. (eds.) *ICATPN 1995*. LNCS, vol. 935, pp. 374–391. Springer, Heidelberg (1995)
5. Ciardo, G., Marmorstein, R., Siminiceanu, R.: Saturation unbound. In: Garavel, H., Hatcliff, J. (eds.) *TACAS 2003*. LNCS, vol. 2619, pp. 379–393. Springer, Heidelberg (2003)
6. Holzmann, G., Smith, M.: A practical method for verifying event-driven software. In: *ICSE 1999: Proceedings of the 21st international conference on Software engineering*, Los Alamitos, CA, USA, pp. 597–607. IEEE Computer Society Press, Los Alamitos (1999)
7. LIP6/Move: the libDDD environment (2007), <http://www.lip6.fr/libddd>
8. Couvreur, J.M., Encrenaz, E., Paviot-Adet, E., Poitrenaud, D., Wacrenier, P.A.: Data Decision Diagrams for Petri Net Analysis. In: Esparza, J., Lakos, C.A. (eds.) *ICATPN 2002*. LNCS, vol. 2360, pp. 1–101. Springer, Heidelberg (2002)
9. Couvreur, J.M., Thierry-Mieg, Y.: Hierarchical Decision Diagrams to Exploit Model Structure. In: Wang, F. (ed.) *FORTE 2005*. LNCS, vol. 3731, pp. 443–457. Springer, Heidelberg (2005)
10. Wang, F.: Formal verification of timed systems: A survey and perspective. *IEEE* 92(8) (2004)
11. Ciardo, G., Siminiceanu, R.: Using edge-valued decision diagrams for symbolic generation of shortest paths. In: Aagaard, M.D., O’Leary, J.W. (eds.) *FMCAD 2002*. LNCS, vol. 2517, pp. 256–273. Springer, Heidelberg (2002)
12. Somenzi, F.: CUDD: CU Decision Diagram Package (release 2.4.1) (2005), <http://vlsi.colorado.edu/fabio/CUDD/cuddIntro.html>
13. Lind-Nielsen, J., Mishchenko, A., Behrmann, G., Hulgaard, H., Andersen, H.R., Lichtenberg, J., Larsen, K., Soranzo, N., Bjorner, N., Duret-Lutz, A., Cohen, H.a.: buddy - library for binary decision diagrams (release 2.4) (2004), <http://buddy.wiki.sourceforge.net/>
14. Ciardo, G.: Reachability Set Generation for Petri Nets: Can Brute Force Be Smart? *Applications and Theory of Petri Nets 2004*, pp. 17–34 (2004)
15. Cabac, L., Duvigneau, M., Moldt, D., Rölke, H.: Modeling Dynamic Architectures Using Nets-within-Nets. In: Ciardo, G., Darondeau, P. (eds.) *ICATPN 2005*. LNCS, vol. 3536, pp. 148–167. Springer, Heidelberg (2005)
16. Biberstein, O., Buchs, D., Guelfi, N.: Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism (2001)

Performance Evaluation of Workflows Using Continuous Petri Nets with Interval Firing Speeds

Kunihiko Hiraishi

School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Nomi-shi, Ishikawa, 923-1292, Japan
hira@jaist.ac.jp

Abstract. In this paper, we study performance evaluation of workflow-based information systems. Because of state space explosion, analysis by stochastic models, such as stochastic Petri nets and queuing models, is not suitable for workflow systems in which a large number of flow instances run concurrently. We use fluid-flow approximation technique to overcome this difficulty. In the proposed method, GSPN (Generalized Stochastic Petri Nets) models representing workflows are approximated by a class of timed continuous Petri nets, called routing timed continuous Petri nets (RTCPN). In RTCPN models, each discrete set is approximated by a continuous region on a real-valued vector space, and variance in probability distribution is replaced with a real-valued interval. Next we derive piecewise linear systems from RTCPN models, and use interval methods to compute guaranteed enclosures for state variables. As a case study, we solve an optimal resource assignment problem for a paper review process.

1 Introduction

How to make information systems safe, dependable and trustworthy is one of central concerns in the development of e-Society and e-Government. There are two kinds of correctnesses, qualitative correctness and quantitative correctness, where the former means that the system is proved to be logically correct, and the latter means that the system has sufficient performance for an expected amount of transactions. In this paper, we focus on quantitative correctness, and study performance evaluation of workflow-based information systems, particularly for those in which many instances of flows run concurrently. Such a situation often arises in web-based information systems for enterprises and local governments.

There are various results on modeling and verification of workflows (e.g., [21,8]). In most of previous researches, some specific formalism, such as Petri nets, UML activity diagrams, and business process modeling languages (e.g., BPMN [22]), are used for describing workflows, and properties such as liveness, soundness and more general properties described by logical formulas are verified by using verification techniques developed for each formalism.

On the other hand, quantitative correctness was also studied as an important issue [5,1,15]. In actual workflow-based information systems, each workflow is nothing but a template, and many instances of the same workflow run concurrently. Therefore, guaranteeing correctness of an individual instance is insufficient for guaranteeing quantitative correctness of the entire system. In this paper, we study an optimal resource assignment problem as one of problems that particularly arise in workflow-based information systems. Recent workflow systems are often used for integrating various resources (software subsystems, databases, workers, machines, other organizations, etc.) that exist in enterprises. Therefore, we need to take care of quantity of resources necessary for performing workflows. Otherwise, many tasks may be assigned to the same resource, and as a result, the resource becomes a bottleneck of the system. To analyze this problem, we can use performance models such as stochastic Petri nets and queuing networks. However, state space explosion prevents us from dealing with a large number of flow instances.

Fluidification (or continuization) is a relaxation technique that tackles the state space explosion by removing the integrality constraints [20]. This idea is not new, and is found in various formalisms such as queuing networks (e.g., [14,17]) and Petri nets. For Petri nets, fluidification was firstly introduced into the model called continuous Petri nets, and the formalism was extended to hybrid Petri nets [4]. Similar idea was also introduced into stochastic models such as fluid stochastic Petri nets [12]. Fluidification was also applied to analysis of a performance model based on process algebra [10].

In this paper, we introduce a class of timed continuous Petri nets, called routing timed continuous Petri nets (RTCPN), in order to approximate discrete state spaces of generalized stochastic Petri nets (GSPN). To deal with variance in probability distribution of each firing delay, interval firing speeds are introduced to timed transitions of RTCPN. In this sense, approximated models have uncertainty in their parameters.

There are several results on analysis of timed continuous Petri nets. In [9,13], linear programming is used for computing performance measures at steady state. In contrast with previous works, we focus on transient analysis. Moreover, since our aim is to prove quantitative correctness, we would like to give some amount of guarantee to obtained results. It is known that the behavior of a timed continuous Petri net is represented by a piecewise linear (PWL) system. Based on interval methods for ordinary differential equations [18,7], we derive a procedure to compute a guaranteed enclosure for state variables at each time step, where the guaranteed enclosure means that true value is surely in the enclosure. All the computation can be performed by linear programming solver and interval arithmetic.

As a case study, we consider the paper review process in an academic journal. The problem is to compute the minimum number of associate editors sufficient for handling an expected amount of submitted papers. We first show a result by GSPN. Next we compute transient behavior of the approximated RTCPN

model, and compare the two results. Consequently, we claim that the proposed approach is scalable for the number of flow instances.

The paper is organized as follows. In Section 2, analysis by GSPN is presented. In Section 3, RTCPN is introduced, and the GSPN model built in Section 2 is approximated by RTCPN. Moreover, we derive a PWL system from the RTCPN model. In Section 4, an interval approximation algorithm for computing transient behavior of PWL systems is described. Numerical results of the algorithm are also shown. Section 5 is the concluding remarks.

2 Modeling of Workflows by Stochastic Petri Nets

2.1 Example: Paper Review Process

We study the following workflow as an example. It is a workflow of the review process of an academic journal. The initial fragment of the workflow is shown in Fig. 1. By the editor in chief, each submitted paper is firstly assigned to one of associate editors responsible for research categories suitable for the paper. Then the associate editor finds two reviewers through some negotiation processes. There are three cases at the first review: acceptance, rejection, and conditional acceptance. If the decision is conditional acceptance, then the associate editor requests the authors to submit a revised manuscript toward the second review. The decision at the second review is the final one, and is either acceptance or rejection.

All the processes are supported by a web-based information system including electronic submission and automatic notification of due dates. Then the problem is how to decide the appropriate number of associate editors in each research category, considering load balancing.

The problem is formally stated as follows:

Given

- A description of workflow,
- Statistics on paper submission,
- An upper bound of the number of papers each associate editor can handle,

Find

- The minimum number of associate editors such that the workflow runs stably. (The number of papers waiting for being processed should not become too large.)

The statistics on paper submission/handling is given as follows:

- Duration between submission and final judgment:
 - Acceptance at the first review: 2.4 months
 - Rejection at the first review: 3.9 months
 - Acceptance at the second review: 5.9 months
 - Rejection at the second review: 6.8 months
- Probabilities of acceptance and rejection:
 - Acceptance at the first review: 0.065

- Rejection at the first review: 0.687
- Acceptance at the second review: 0.238
- Rejection at the second review: 0.010
- Average number of paper submissions: 16.9/month.

(This statistics is obtained from actual data of some academic journal.)

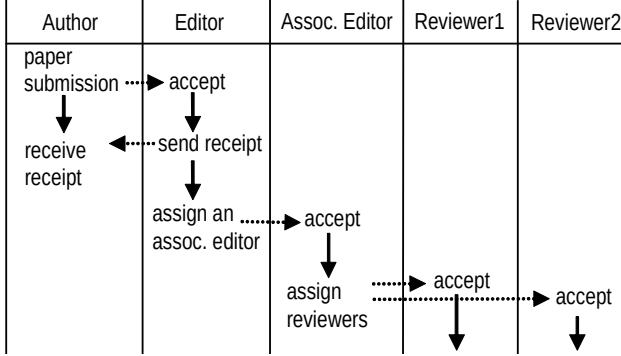


Fig. 1. The paper review process of an academic journal

2.2 Stochastic Petri Nets

A *generalized stochastic Petri net* (GSPN) [Ajmone Marsan95] is a 6-tuple $GSPN = (P, T, A, m_0, \lambda, w)$, where P is a set of places, T is a set of transitions, $A : P \times T \cup T \times P \rightarrow \mathbb{N}$ is the incidence function that specifies the weights of the arcs between places and transitions, and $m_0 : P \rightarrow \mathbb{N}$ is the initial marking. The incidence function is equivalently represented by two nonnegative integer matrices $\mathbf{A}^+ = [a_{ij}^+]$, $\mathbf{A}^- = [a_{ij}^-] \in \mathbb{N}^{|P| \times |T|}$ by $a_{ij}^+ = A(t_j, p_i)$ and $a_{ij}^- = A(p_i, t_j)$. Let $\mathbf{A} = \mathbf{A}^+ - \mathbf{A}^-$ be called the *incidence matrix*.

Transitions of GSPN are partitioned into two different classes: *immediate transitions* and *timed transitions*. Immediate transitions fire in zero time, and timed transitions fire after a random, exponentially distributed, enabling time. The function $\lambda : T_{exp} \rightarrow \mathbb{R}^+$ assigns a firing rate to each timed transition, where $\mathbb{R}^+$ is the set of nonnegative real numbers. The function $w : T_{im} \rightarrow \mathbb{R}^+$ assigns a firing weight to each immediate transition.

Firing semantics is described as follows. If the set of enabled transitions H comprises only timed transitions, then each transition $t_j \in H$ fires with probability $\lambda(t_j) / \sum_{t_k \in H} \lambda(t_k)$. If H comprises both immediate and timed transitions, then only immediate transitions can fire. If H has several conflicting immediate transitions, then the firing weights determine probability that a particular immediate transition will fire. Let $C \subseteq H$ be the set of conflicting immediate transitions. Then the probability, called switching probability, that each transition $t_j \in C$ will fire is $w(t_j) / \sum_{t_k \in C} w(t_k)$.

The firing semantics described above is called *finite-server semantics*. There is another firing semantics, called *infinite-server semantics*. In infinite-server semantics, the same transition fires simultaneously. The multiplicity in the firing depends on the number of tokens in input places of each transition. As usual, we use the notation $\bullet x := \{y \in P \cup T \mid A(y, x) > 0\}$ and $x^\bullet := \{y \in P \cup T \mid A(x, y) > 0\}$. For an infinite-server timed transition t_j and a marking m , let $enab(t_j, m) := \min_{p_i \in \bullet t_j} \{m(p_i)/A(p_i, t_j)\}$. Then the marking-dependent firing rate of t_j at marking m is $\lambda(t_j) \cdot enab(t_j, m)$. We will allow both types of timed transitions to exist in a model.

We first use GSPN to compute performance measures such as the average number of papers waiting for being processed. One of advantages of using Petri net models is that each instance of workflows is modeled by a token. Increasing the number of workflow instances corresponds to increasing the number of tokens. This means that modeling by GSPN is scalable in the number of workflow instances. Using analysis techniques on GSPN, we can compute stationary probability distribution of reachable states.

Fig. 2 is the GSPN model of the workflow, where research categories of associate editors are not considered for simplicity. The number associated with each transition is the firing delay (if the transition is timed) or the firing weight (if the transition is immediate). These numbers are determined from the statistics on paper submission/handling.

Once a paper is submitted, i.e., transition ‘submit’ fires, the paper is waiting for being processed. If an associate editor is available, then the result of the first review is decided according to the given probability. If the result is conditional acceptance, then the paper proceeds to the second review, and the final result is decided according to the given probability. These probabilities are specified by weights of immediate transitions. In the GSPN model, there are two sets of potentially conflicting immediate transitions: one corresponds to the decision at the first review, and the other corresponds to the decision at the second review.

Each delay τ in the paper review workflow is given as a firing rate $1/\tau$ of the corresponding timed transition. Since papers are processed in parallel by associate editors, these timed transitions are defined to be infinite-server (as indicated by symbol ∞ in the figure). Only transition ‘submit’ is a single-server timed transition. The firing rate ‘1’ of transition ‘submit’ specifies the average number of papers added to place ‘waiting papers’ per one unit of time (= month). Since the average number of paper submissions is 16.9/month, one token corresponds to 16.9 papers.

The place ‘paper pool’ is necessary for the state space to be finite, and needs to be nonempty in order to get a correct result. Otherwise, transition ‘submit’ does not fire with the defined rate. As long as ‘paper pool’ is nonempty, we do not have to care about the number of tokens in it. The return arc to place ‘submit’ is introduced just for keeping the number of papers in the system constant. At the initial marking, we put a sufficient number of tokens in place ‘paper pool’. In computer experiments, we check probability that the number of tokens in ‘paper pool’ is 0.

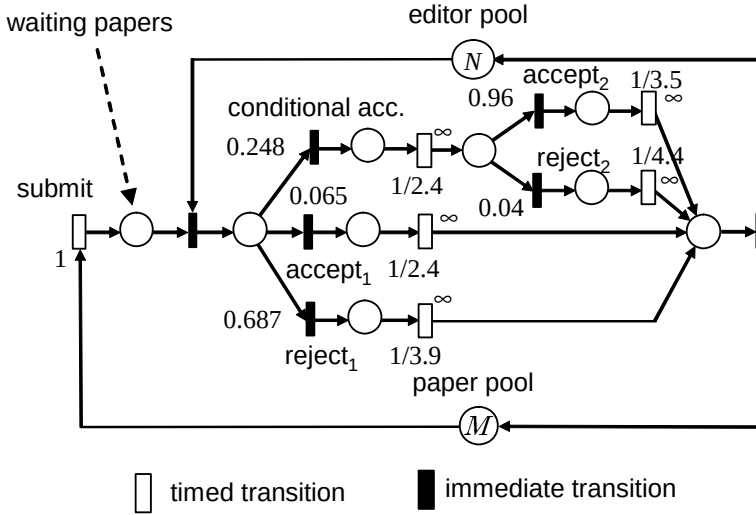


Fig. 2. GSPN model

2.3 Computation Results: GSPN Model

We compute the expected number of waiting papers at steady state by a computer tool DSPNexpress-NG [16,23]. Since exponential distribution is not appropriate for the delay of each transition, we use Erlangian distribution of order 2 as the probability distribution of each timed transition. It is known that Erlangian distribution of order n with average rate λ is simulated by a serial connection of n exponentially-distributed timed transitions with average rate $n\lambda$ [6]. Using this technique, we introduce Erlangian-distributed timed transitions in GSPN models. The drawback to using this technique is that it increases the size of the state space. Considering the actual probability distribution, order 2 may be insufficient. In computer experiments, however, it was hard to compute solutions for models with Erlangian distribution of order more than 2, because of state space explosion.

The computation result is shown in Table 1. In the initial state, we put N tokens in place 'editor pool'. For each value of N , the number of tangible states, CPU time, the expected number of waiting papers at steady state, and probability that the paper pool is empty are shown in Table 1. The computer environment used for the computation is a Linux high-performance computer with Intel Itanium2, 1.6GHz / 9MB Cache CPU, 16GB main memory. It is observed that $N \geq 6$ gives a desirable level. The amount of resources $N = 6$ means that $6 \times 16.9 = 101.4$ papers can be processed in parallel.

A high probability of $p(\#paperpool = 0)$ means that the amount of resources is insufficient for handling papers. If the probability is high, most of tokens initially given in the place 'paper pool' are stuck in place 'waiting papers'. We also observe that CPU time and the size of the state space increase very rapidly.

Table 1. Numerical results of GSPN analysis

N	#states	CPU Time (sec.)	#waiting papers	$p(\#paperpool = 0)$
3	2926	0.31	0.18	0.30
4	8866	0.7	5.94	0.094
5	23023	2.3	1.99	0.013
6	53053	6.2	0.63	0.0021
7	110968	15	0.21	0.00049
8	213928	29	0.08	0.00020
9	384098	58	0.03	0.00010

3 Modeling by a Class of Timed Continuous Petri Nets

In the GSPN model (Fig. 2), the number of reachable states increases exponentially in the number N . As a result, we will not be able to compute the steady state for larger models. We will use fluidification technique to overcome this difficulty. For this purpose, we introduce a class of timed continuous Petri nets as formalism to deal with continuous dynamics.

3.1 Routing Timed Continuous Petri Nets

We define a class of timed continuous Petri nets that will be used for approximating the GSPN model. A *routing timed continuous Petri nets* (RTCPN) is a 6-tuple $RTCPN = (P, T, A, m_0, \lambda, w)$, where P , T , and A are the same as those in GSPN except that the range of A is the set of real numbers, and $m_0 : P \rightarrow \mathbb{R}^+$ is the initial marking.

Transitions of RTCPN are partitioned into *timed transitions* and *routing transitions*. Timed transitions correspond to infinite-server timed transitions of GSPN, and routing transitions correspond to immediate transitions of GSPN.

In RTCPN, we put the following restriction on the Petri net structure. RTCPN still can represent FORK-JOIN type structure of workflows under these restrictions.

- If a place p_i has an output routing transition, then every output transition of p_i is a routing transition.
- For any two routing transition t_i and t_j , if $\bullet t_i \cap \bullet t_j \neq \emptyset$ then $|\bullet t_i| = |\bullet t_j| = 1$, i.e., every potentially conflicting routing transition has a *free-choice structure*.
- There is no cycle consisting only of routing transitions.

The function $\lambda : T_{time} \rightarrow \mathbb{R}^+$, where T_{time} is the set of timed transitions, assigns a firing speed to each timed transition. A timed transition t_j fires at speed $\lambda(t_j) \cdot enab(t_j, m)$, similarly to an infinite-server timed transition of GSPN.

Routing transitions together with the routing rate function $w : T_{route} \rightarrow \mathbb{R}^+$, where T_{route} is the set of routing transitions, are used for determining static routing of flows. This idea is similar to one in STAR-CPN [Gauj04].

Let t_j be a non-conflicting routing transition. There are two types in firing:

- (i) When $enab(t_j, m) > 0$, a quantity $A(p_i, t_j) \cdot enab(t_j, m)$ is removed from each input place p_i of t_j , and a quantity $A(t_j, p_i) \cdot enab(t_j, m)$ is added to each output place p_i of t_j . These actions are instantaneous, and as a result, at least one of input transitions of t_j becomes empty (see Fig. 3). This case may occur only at the initial marking, or at the time when external quantity is put on a place.
- (ii) Suppose that $enab(t_j, m) = 0$ and every empty input place p_i is fed, i.e., a positive quantity V_i^+ flows into p_i . Let E be the set of empty input places of p_i , and we define the firing degree by $deg(t_j, m) := \min_{p_i \in E} \{V_i^+ / A(p_i, t_j)\}$. Then a quantity $A(p_i, t_j) \cdot deg(t_j, m)$ flows from each input place p_i of t_j , and a quantity $A(t_j, p_k) \cdot deg(t_j, m)$ flows into each output place p_k of t_j (see Fig. 4). We remark that nonempty input places are irrelevant to the firing degree.

For simplicity, we assume that in the initial marking m_0 , $enab(t_j, m_0) = 0$ holds for every routing transition t_j , and ignore the type (i) firing. In the type (ii) firing, which one of input places determines the firing degree may be switched during execution.

The firing rule for conflicting routing transitions is as follows. Let C be the set of routing transitions that have a common input place p_i . Then a fraction $r(t_j) := w(t_j) / \sum_{t_k \in C} w(t_k)$ of flow V_i^+ is used exclusively for firing of each $t_j \in C$. Let $deg(t_j, m) := r(t_j) \cdot V_i^+ / A(p_i, t_j)$. Then, similarly to the non-conflicting case, a quantity $A(p_i, t_j) \cdot deg(t_j, m)$ flows from each input place p_i of t_j , and a quantity $A(t_j, p_k) \cdot deg(t_j, m)$ flows into each output place p_k of t_j (see Fig. 5).

3.2 Approximating Probability Distributions

From the GSPN model, we build an approximated RTCPN model based on the following idea:

- An infinite-server timed transition t_j of GSPN (with mean firing rate λ_j) is approximated by a timed transition with a marking dependent firing speed $\lambda_j \cdot enab(t_j, m)$.

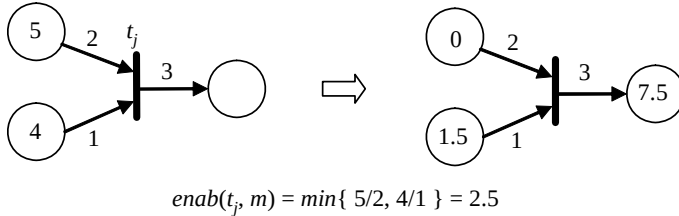


Fig. 3. Type (i) firing of a routing transition

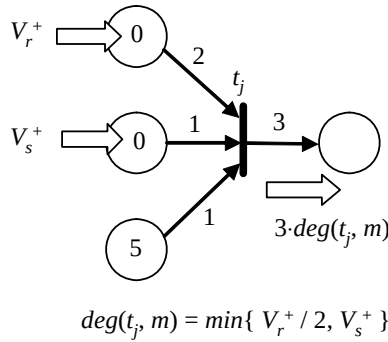


Fig. 4. Type (ii) firing of a routing transition (non-conflicting case)

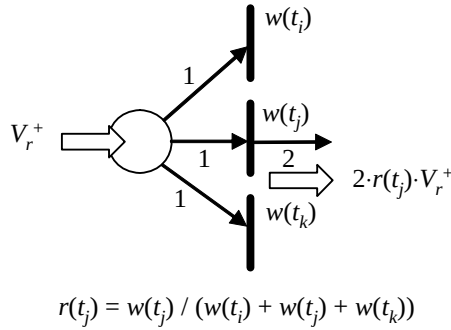


Fig. 5. Type (ii) firing of a routing transition (conflicting case)

- The variance of the probability distribution for firing rate λ_j is approximated by an interval firing speed $[\lambda_j] = [\underline{\lambda}_j, \overline{\lambda}_j]$. We assume that the actual firing delay is within the interval with a high probability.
- Switching probability among conflicting immediate transitions is approximated by routing rates of routing transitions.

Remark 1. (1) Approximation by mean firing rates preserves the expected number of tokens in each place of GSPN. We consider the following simple situation. Let t_j be an infinite-server timed transition with firing rate λ_j in GSPN. Suppose that t_j is non-conflicting and has a unique input place p_i with arc weight 1. Let W_i denote the random variable representing firing delay of t_j . Suppose that at time 0, there are k tokens in p_i . Some amount of time $t > 0$ has elapsed and the remaining tokens has decreased to $k' < k$ by firing of t_j . Then the expected value of k' is $E[m(p_i)] = kP[W_j > t] = ke^{-\lambda_j t}$ since t_j is exponentially distributed in GSPN. In the RTCPN model, the firing speed of the corresponding transition is $\lambda_j x_i$, where x_i is the variable representing $m(p_i)$. Solving the differential equation with initial condition $x_i(0) = k$, we obtain $x_i(t) = ke^{-\lambda_j t}$. Both values coincide. This holds in general cases by the linearity of the system.

(2) For “congested” systems, this approximation is valid for any probabilistic distribution [20]. This is a result of the central limit theorem. Suppose that in some stochastic Petri net, delay for single firing has any probabilistic density function with mean τ and variance σ^2 . Then mean delay for firing with multiplicity n is approximately normally distributed with the same mean τ and variance σ^2/n , provided that the time instant at which each of n firings becomes enabled is randomly selected. Considering this fact, the size of the interval $[\underline{\lambda}_j, \overline{\lambda}_j]$ can be narrowed to one proportional to $1/\sqrt{enab(t_j, m)}$, when the system is approaching to a steady state.

From the GSPN model in Fig. 2, we obtain an RTCPN model shown in Fig. 6. There is no ‘paper pool’ in the RTCPN model since we do not have to make the state space finite any more. Since RTCPN has no timed transitions corresponding to single-server transitions, a single-server transition ‘submit’ is simulated by a timed transition with a self-loop.

3.3 From RTCPN Model to Differential Equations

We can derive a set of differential equations from the RTCPN model. We first define the flow for arc $A(p_i, t_j)$ by $V(p_i, t_j) := A(p_i, t_j) \cdot \lambda(t_j) \cdot enab(t_j, m)$ if t_j is a timed transition; $V(p_i, t_j) := A(p_i, t_j) \cdot deg(t_j, m)$ if t_j is a routing transition (type (ii) firing). Similarly, the flow of arc $A(t_j, p_i)$ is $V(t_j, p_i) := A(t_j, p_i) \cdot \lambda(t_j) \cdot enab(t_j, m)$ if t_j is a timed transition; $V(t_j, p_i) := A(t_j, p_i) \cdot deg(t_j, m)$ if t_j is a routing transition (type (ii) firing).

If p_i has no output routing transitions, then the differential equation with respect to place p_i is

$$\dot{m}(p_i) = V_i^+ - V_i^- \quad (1)$$

where

$$V_i^+ = \sum_{t_j \in \bullet p_i} V(t_j, p_i), \quad V_i^- = \sum_{t_j \in p_i^*} V(p_i, t_j) \quad (2)$$

The differential equation (1) is not linear in general because $enab(t_j, m)$ and $deg(t_j, m)$ contain ‘min’ operator. However, we can rewrite the set of differential equations as a piecewise linear (PWL) system.

Theorem 1. *For the set of differential equations (1) derived from RTCPN, there exists an equivalent PWL system.*

Proof. We first remark that each marking $m : P \rightarrow \mathbb{R}$ can be identified with a real vector $m \in \mathbb{R}^{|P|}$. For $enab(t_j, m)$, where t_j is a timed transition, we define a polytope

$$\xi_{time}[t_j, p_k] := \{m \in \mathbb{R}^{|P|} \mid \forall p_i \in \bullet t_j - \{p_k\}. m(p_k)/A(p_k, t_j) \leq m(p_i)/A(p_i, t_j)\}.$$

Then $enab(t_j, m)$ is replaced with $m(p_k)/A(p_k, t_j)$ under the condition $m \in \xi_{time}[t_j, p_k]$.

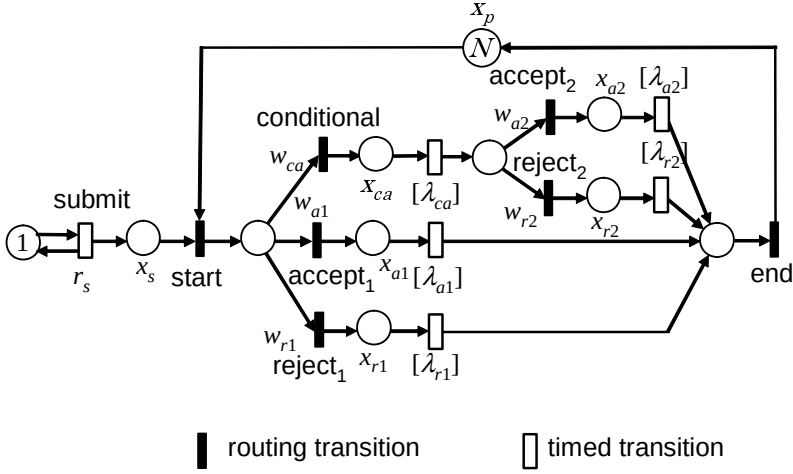


Fig. 6. RTCPN model

Similarly for a non-conflicting routing transition t_j , we define the following set, which is not necessarily a polytope:

$$\xi_{route}[t_j, p_k, E] := \{m \in \mathbb{R}^{|P|} \mid [\forall p_i \in E. m(p_i) = 0] \wedge [\forall p_i \in \bullet t_j - E. m(p_i) > 0] \wedge [\forall p_i \in E - \{p_k\}. V_k^+(m)/A(p_k, t_j) \leq V_i^+(m)/A(p_i, t_j)]\},$$

where $E \subseteq \bullet t_j$ and $V_i^+(m)$ is the total flow to place p_i determined by the current marking m . Suppose that all transitions in $\bullet p_i$ are timed transitions. Note that if at least one of input transitions is a routing transition, then all other input transitions are also routing transitions as we have assumed. Then $V_i^+(m)$ is represented by a piecewise linear function of m . This is a result of the partition by ξ_{time} 's. Therefore, there exists a finite set of polytopes $\{\xi_{route}^l[t_j, p_k, E]\}$ such that we can replace $deg(t_j, m)$ with $V_k^+/A(p_k, t_j)$ when $m \in \xi_{route}^l[t_j, p_k, E]$. Moreover, if such a finite set of polytopes are obtained for every input transition, which is not necessarily a timed transition, of place p_i , then $V_i^+(m)$ is represented by a piecewise linear function. Since there is no cycle consisting only of routing transitions as we have assumed, we can obtain such polytopes for all transitions.

Using the finite partition of $\mathbb{R}^{|P|}$ obtained by the above polytopes, we can have an equivalent PWL system. $\square$

In the RTCPN model in Fig. 6, we assign a state variable x_i to each place, as indicated in the figure. Then we obtain a PWL system with two regions shown in Fig. 7. There are two modes in the PWL system. At least one of x_p and x_s is empty at every time instant.

4 Guaranteed Transient Analysis by Interval Method

In this section, we show a method for transient analysis of PWL systems derived from RTCPN models. The result includes guaranteed enclosures of state

Mode I: $\{x_p = 0, x_s > 0\}$	Mode II: $\{x_p > 0, x_s = 0\}$
$\dot{x}_s = r_s - R(x)$	$\dot{x}_s = 0$
$\dot{x}_p = 0$	$\dot{x}_p = R(x) - r_s$
$\dot{x}_{ca} = p_{ca}R(x) - [\lambda_{ca}]x_{ca}$	$\dot{x}_{ca} = p_{ca}r_s - [\lambda_{ca}]x_{ca}$
$\dot{x}_{a1} = p_{a1}R(x) - [\lambda_{a1}]x_{a1}$	$\dot{x}_{a1} = p_{a1}r_s - [\lambda_{a1}]x_{a1}$
$\dot{x}_{r1} = p_{r1}R(x) - [\lambda_{r1}]x_{r1}$	$\dot{x}_{r1} = p_{r1}r_s - [\lambda_{r1}]x_{r1}$
$\dot{x}_{a2} = p_{a2}[\lambda_{ca}]x_{ca} - [\lambda_{a2}]x_{a2}$	$\dot{x}_{a2} = p_{a2}[\lambda_{ca}]x_{ca} - [\lambda_{a2}]x_{a2}$
$\dot{x}_{r2} = p_{r2}[\lambda_{ca}]x_{ca} - [\lambda_{r2}]x_{r2}$	$\dot{x}_{r2} = p_{r2}[\lambda_{ca}]x_{ca} - [\lambda_{r2}]x_{r2}$

$$R(x) = [\lambda_{a1}]x_{a1} + [\lambda_{r1}]x_{r1} + [\lambda_{a2}]x_{a2} + [\lambda_{r2}]x_{r2}.$$
Fig. 7. The PWL system derived from the RTCPN model

variables. The method is based on interval methods for ordinary differential equations. All the computation can be performed by interval arithmetic and linear programming.

4.1 Interval Method

Interval methods for ordinary differential equation (ODE) systems were introduced by Moore [18]. These methods provide numerically reliable enclosures of the exact solution at discrete time points $t_0, t_1, \dots, t_k$.

We consider the following nonlinear continuous-time system:

$$\dot{x}(t) = f(x(t), \theta), \quad x(t_0) = x_0 \quad (3)$$

where $x \in \mathbb{R}^n$ and θ is the vector of uncertain system parameters. Each parameter θ_i is assumed to be bonded by an interval $[\underline{\theta}_i, \overline{\theta}_i]$. In what follows, we will write $[v]$ to denote a vector of intervals that give the range of each component of a vector v . First the continuous-time state equation (3) is discretized by Taylor series expansion with respect to time:

$$x(t_{k+1}) = x(t_k) + \sum_{r=1}^{\tau} \frac{h^r}{r!} f^{(r-1)}(x(t_k), \theta) + e(x(\eta), \theta) \quad (4)$$

with $h = t_{k+1} - t_k$ and $t_k \leq \eta \leq t_{k+1}$. The guaranteed bound for the time discretization error is calculated by

$$e(x(\eta), \theta) \subseteq [e_k] = \frac{h^{\tau+1}}{(\tau+1)!} F^{(\tau)}([B_k], [\theta]) \quad (5)$$

where $F^{(\tau)}$ is the interval extension of the τ -th order derivative $f^{(\tau)}$ of f , and $[B_k]$ is the bounding box for the range of state variables $x(t) \in [B_k]$, $\forall t \in [t_k, t_{k+1}]$. The bounding box $[B_k]$ is computed by applying the Picard operator

$$\Phi([B_k]) := [x_k] + [0, h] \cdot F([B_k], [\theta]) \subseteq [B_k] \quad (6)$$

where F is the interval extension of the function f . Calculation is usually performed by interval arithmetic. The bounding box $[B_k]$ is initialized with the state vector $[x(t_k)]$. If $\Phi([B_k]) \not\subseteq [B_k]$, then the bounding box $[B_k]$ has to be enlarged. If $\Phi([B_k]) \subseteq [B_k]$, then (6) is evaluated recursively until the deviation between $\Phi([B_k])$ and $[B_k]$ is smaller than a specified value. In the case that this algorithm does not converge or that the interval of the discretization error $[e_k]$ is unacceptably large, the step size h has to be reduced.

There are several ways to compute a guaranteed bound $[x(t_{k+1})]$ by equations (4) and (5) [7]. The direct interval technique is to compute $[x(t_{k+1})]$ by interval arithmetic, i.e.,

$$[x(t_{k+1})] = [x(t_k)] + \sum_{r=1}^{\tau} \frac{h^r}{r!} F^{(r-1)}([x(t_k)], [\theta]) + [e_k] \quad (7)$$

The direct interval techniques propagate entire boxes through interval solutions. As a consequence errors may tend to accumulate as computations proceed. Let $v|_i$ denote the i -th component of a vector v . In the piecewise interval technique, the solution is computed by

$$[x(t_{k+1})]|_i = \left[\inf_{x \in [x(t_k)], \theta \in [\theta]} \psi(x, \theta)|_i, \sup_{x \in [x(t_k)], \theta \in [\theta]} \psi(x, \theta)|_i \right] + [e_k]|_i \quad (8)$$

where $\psi(x, \theta) := x + \sum_{r=1}^{\tau} (h^r/r!) f^{(r-1)}(x, \theta)$, i.e., intervals are computed by solving optimization problems for each component. The main idea of the piecewise interval technique is to propagate small boxes, and works effectively for reducing the accumulation of error.

4.2 Piecewise Interval Method for Systems with Multiple Modes

For systems with multiple modes like PWL systems, we need to take account of mode changes during time interval $[t_k, t_{k+1}]$. Such systems with switching are studied in [19]. We briefly describe the interval method for a PWL system

$$\dot{x}(t) = f_j(x(t), \theta) \text{ if } x \in \xi_j \quad (9)$$

where $x \in \mathbb{R}^n$, each f_j represents a linear continuous-time dynamic with a vector of interval parameters θ , and each ξ_j is a polytope in $\mathbb{R}^n$. The idea is to replace function f in equation (3) with the following function f_a that represents the union of f_j 's for all active modes during time interval $[t_k, t_{k+1}]$, i.e.,

$$F_{f_a} \supseteq \bigcup_{j \in M([B_k])} F_{f_j} \quad (10)$$

where $[B_k]$ is a bounding box for time interval $[t_k, t_{k+1}]$, $M([B_k])$ denote the set of all mode j such that $\xi_j \cap [B_k] \neq \emptyset$, and F_f represents the exact value set

$$F_f := \{y \mid y = f([B_k], [\theta])\} \quad (11)$$

of function f under consideration of all interval arguments. Since the bounding box $[B_k]$ and function f_a are mutually dependent, we need to perform an iterative procedure to compute them.

We use here a piecewise interval technique which was not used in [19]. This is possible because we are considering only linear systems. Considering that the system is piecewise linear, we will use $\tau = 1$ in equations (4) and (5).

The result by applying Picard operator is computed by the following piecewise computation:

$$\Phi(B_k)|_i = \left[\inf_{(j,x,y,d,\theta) \in S_1^k} \phi_j(x,\theta)|_i, \sup_{(j,x,y,d,\theta) \in S_1^k} \phi_j(x,\theta)|_i \right] \quad (12)$$

where

$$S_1^k := \{(j, x, y, d, \theta) \mid j \in M([B_k]), x \in [x(t_k)] \cap \Theta, y \in [B_k] \cap \Theta, d \in [0, h], \theta \in [\theta]\} \quad (13)$$

and $\phi_j(x, y, d, \theta) := x + d \cdot f_j(y, \theta)$ for each region ξ_j . Θ is the additional linear constraints that we can assume on the state space. As the constraints Θ , we will use P-invariants derived from RTCPN models later. Let *BoundingBox* $([x(t_k)], [\theta], \Theta, h)$ denote the above procedure to compute the bounding box.

The piecewise interval method to compute guaranteed bounds for $x(t_k)$, $k = 0, 1, \dots, k_f$ is shown in Fig. 8, where piecewise computation is used at the step (*) for computing $[x(t_{k+1})]_i$.

solve $([x(t_0)], [\theta], \Theta, t_0, h, k_f) ::$

for $k = 0$ **to** $k_f - 1$ **do**

$[B_k] := \text{BoundingBox}([x(t_k)], [\theta], \Theta, h);$

$[e_k] := \frac{h^2}{2!} F^{(1)}([B_k], [\theta]);$

$[x(t_{k+1})]_i := \left[\inf_{(j,x,d) \in S_2^k} \psi_j(x,\theta)|_i, \sup_{(j,x,d) \in S_2^k} \psi_j(x,\theta)|_i \right] + [e_k]_i \text{ (*)}$

where

$S_2^k := \{(j, x, \theta) \mid j \in M([B_k]), x \in [x(t_k)] \cap \Theta, \theta \in [\theta]\}$ and
 $\psi_j(x, \theta) := x + h \cdot f_j(x, \theta);$

endfor.

Fig. 8. Procedure to compute guaranteed enclosures

Theorem 2. *Suppose that the PWL system (9) satisfies the following condition: for any bounded set $X \subseteq \mathbb{R}^n$, the number of regions ξ_j such that $X \cap \xi_j \neq \emptyset$ is finite. Then the computation of (12) and step (*) in procedure *solve()* can be performed by solving a finite number of linear programming (LP) problems.*

Proof. We first prove the theorem for (12). Constraints for $x \in [x(t_k)] \cap \Theta$, $y \in [B_k] \cap \Theta$, and $\theta \in [\theta]$ are all linear. Let $x = [x_1, \dots, x_n]^T$ and $y = [y_1, \dots, y_n]^T$. Each component of $\phi_j(x, y, d, \theta) = [\phi_{j,1}, \dots, \phi_{j,n}]^T$ has the form $\phi_{j,l} = x_l + d \cdot \sum_i \alpha_i y_i$, where $d \in [0, h]$ and $\alpha_i \in [\underline{\alpha}_i, \overline{\alpha}_i]$. By introducing new variables z_i , $i = 1, \dots, n$, $\phi_{j,l}$ can be rewritten as $\phi_{j,l} = x_l + d \cdot \sum_i z_i$ with linear constraints $\underline{\alpha}_i y_i \leq z_i \leq \overline{\alpha}_i y_i$ if $\underline{\alpha}_i \geq 0$ and $y_i \geq 0$, and constraints for other cases are similarly obtained. Since scalar variable d is constrained only by interval $[0, h]$, $\phi_{j,l}$ gives its optimal value when $d = 0$ or $d = 1$. Therefore, we can compute the interval of $\phi_{j,l}$ by solving LP problems for all $j \in M([B_k])$ and $d \in \{0, h\}$, where $M([B_k])$ is finite by the assumption. The situation is the same in optimization problems of (*) except that d is fixed to h . $\square$

The PWL system obtained from RTCPN satisfies the condition in the above theorem, because the system consists of a finite number of regions. Therefore, the procedures for computing (12) and (*) in *solve()* can be implemented on a constraint solver that can solve linear programming problems. On the other hand, the computation $[e_k]$ includes Jacobian $f^{(1)}$, and we need solve nonlinear constraints in order to compute piecewise interval solutions. By this reason, interval arithmetic is used for computing $[e_k]$.

4.3 Using P-Invariants as Constraints of LP Problems

A P-invariant is a nonnegative integer solution $y \in \mathbb{N}^{|P|}$ of a linear homogeneous equation $y^T \mathbf{A} = 0$, where $\mathbf{A}$ is the incidence matrix of the Petri net structure. As well known, $y^T m = y^T m_0$ holds for any marking m reachable from the initial marking m_0 . Since the marking of RTCPN is defined in a real vector space, we define a P-invariant of RTCPN as any real-valued solution $y \in \mathbb{R}^{|P|}$ of equation $y^T \mathbf{A} = 0$. When we solve optimization problems in (12) and (*), we can add an equation $y^T x = y^T x_0$, where y is a P-invariant and x_0 is the initial state, as one of linear constraints Θ . This will work effectively in reducing the sizes of intervals. In the RTCPN model in Fig. 6, the following P-invariant is obtained:

$$x_p + x_{ca} + x_{a1} + x_{r1} + x_{a2} + x_{r2} = N.$$

Moreover, the following inequalities also hold in all reachable states. We can add these (in)equalities as constraints of the LP problems:

$$x_s \geq 0, x_p \geq 0, x_{ca} \geq 0, x_{a1} \geq 0, x_{r1} \geq 0, x_{a2} \geq 0, x_{r2} \geq 0.$$

4.4 Computation Results: RTCPN Model

As shown in Theorem 2, we need to solve LP problems a number of times. Moreover, interval arithmetic is used in the computation of $[e_k]$. All processes of the algorithm are implemented on a single constraint solver called KCLP-HS, which is a rapid prototyping tool for algorithms on hybrid systems developed by the author’s group [11]. KCLP-HS is a constraint logic programming language equipped with a linear constraint solver, quadratic programming solver, manipulation of convex polyhedra, and interval arithmetic. Disjunctions of linear constraints are handled by backtracking mechanism. We compute at each time step guaranteed intervals for state variables under the following parameters:

- Initial state: $x_p = N$, ($N = 50, 60, \dots, 150$). Values of other state variables are all 0.
- Interval of firing speeds: $[(1-\delta) \cdot \lambda(t_j), (1+\delta) \cdot \lambda(t_j)]$ for each timed transition t_j , where δ is a parameter determining the width of intervals.

Table 2. CPU times and the upper bound of x_s after 12 months ($\delta = 0.2$)

N	CPU Time (sec.)	upper bound of x_s
50	24.0	73.8
60	22.6	49.6
70	19.9	29.3
80	16.4	12.2
90	11.2	1.1
100	7.7	0
110	6.2	0
120	6.2	0
130	6.2	0
140	6.2	0
150	6.2	0

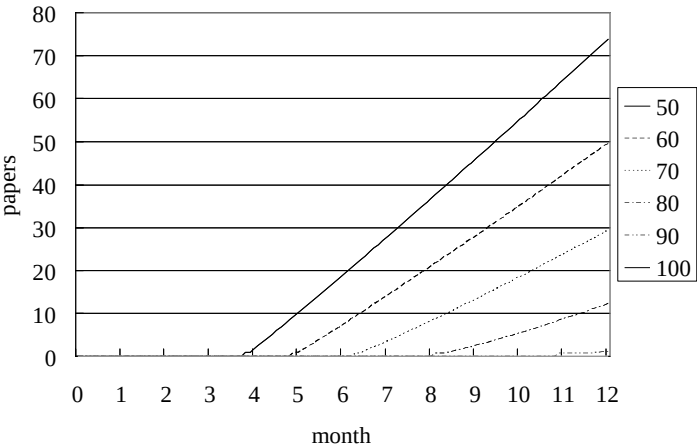


Fig. 9. Upper bounds of x_s ($\delta = 0.2$)

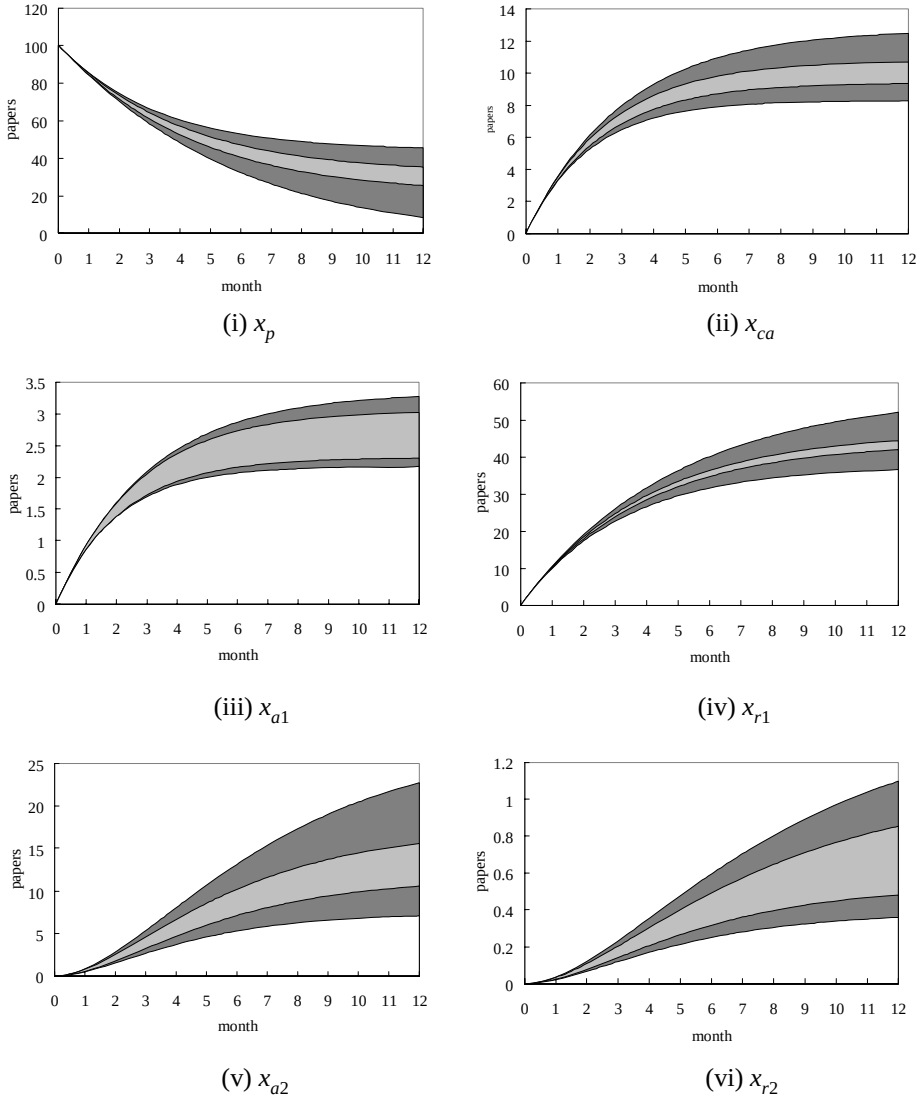


Fig. 10. Guaranteed enclosures of state variables ($N = 100$). (Gray area: $\delta = 0.2$, Light gray area: $\delta = 0.2/\sqrt{\text{enab}(t_j, m)}$).

- Step size: $h = 0.1$ month.
- Duration: 12 months. This means that the number of iterations is 120.

Fig. 9 shows the upper bound of x_s , i.e., the number of waiting papers, for each N , when $\delta = 0.2$. Fig. 10 shows the guaranteed enclosures of state variables in case of $N = 100$, where two enclosures are depicted for each state variable, one is obtained by a fixed $\delta = 0.2$, and the other is obtained by $\delta = 0.2/\sqrt{\text{enab}(t_j, m)}$

as discussed in Remark 1. Table 2 shows CPU times and upper bounds of x_s after 12 months. CPU times are measured by the same computer environment as that used for the GSPN analysis. Note that the value of N does not depend on the computation time. This shows that the method is scalable for the amount of resources.

From these results, we observe that $N > 100$ is sufficient to keep the number of waiting papers at a low level. Looking at the transient behavior in Fig. 10, the quantity x_p , which corresponds to the residual resource, is nearly converged after 12 months. Notice that the number $N = 100$ is almost the same as the number computed by the GSPN model.

5 Concluding Remarks

For performance evaluation of workflows, we have tried two approaches, analysis by GSPN and approximation by a class of timed continuous Petri nets, called RTCPN. A PWL system is derived from the RTCPN model. Using piecewise interval method, guaranteed enclosures for state variables at each time step have been computed. The difference between two approaches is in scalability for the number of workflow instances. Computation times in RTCPN analysis do not depend on the number of workflow instances. Moreover, since state variables in RTCPN are almost decoupled, we expect that interval methods can be applied to larger models including hundreds of variables. Experiments for larger models remain as future work.

From the theoretical point of view, contribution of this paper is summarized as follows. Firstly, we have proposed a new class of continuous Petri net RTCPN, which can be used for fluidification of GSPN models. Secondly, we have proposed to use interval methods for transient analysis of RTCPN models. The method is based on piecewise interval methods for multi-mode systems, and computes guaranteed enclosures of state variables. All computations can be performed by solving a finite number of linear programming problems together with interval arithmetic. Using place invariants in the computation of intervals is also an original idea.

Transient analysis may correspond to bounded model checking for discrete-state systems [3], where the objective of bounded model checking is to find errors by simulation for finite time steps. The models we have studied in this paper are autonomous. We expect that the proposed approach is applicable to performance models with external logical control. Hybrid system models such as hybrid Petri nets can be used for the modeling. In addition, the proposed method is able to check properties of systems with uncertainty in their parameters. Such systems are often found in medical and biological systems. These are also targets of our approach.

Acknowledgments. The research is partly supported by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 17560386, and also 21st Century COE program “Verifiable and Evolvable E-Society” at JAIST. The author thanks to Prof. Hofer and

Mr. Rauh, University of Ulm, Germany, for valuable suggestion about interval methods. The author also thanks to Prof. Lindemann, University of Dortmund, for allowing us to use DSPNexpress-NG.

References

1. Abate, A.F., Esposito, A., Grieco, N., Nota, G.: Workflow Performance Evaluation through WPQL. In: Proc. SEKE 2002, pp. 489–495 (2002)
2. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modeling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. John Wiley and Sons, Chichester (1995)
3. Biere, A., et al.: Bounded Model Checking. Book chapter: Advances in Computers, vol. 58. Academic Press, London (2003)
4. David, R., Alla, H.: On Hybrid Petri Nets. Discrete Event Dynamic Systems: Theory and Applications 11, 9–40 (2001)
5. Dehnert, J., Freiheit, J., Zimmermann, A.: Modeling and Performance Evaluation of Workflow Systems. In: Proc. 4th. World Multiconference on Systems, Cybernetics and Informatics, vol. VIII, pp. 632–637 (2000)
6. Desrochers, A.A., Al-Jaar, R.Y.: Applications of Petri Nets in Manufacturing Systems. IEEE Press, Los Alamitos (1995)
7. Deville, Y., Janssen, M., Hentenryck, P.V.: Consistency Techniques in Ordinary Differential Equations. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 162–176. Springer, Heidelberg (1998)
8. Eshuis, R., Wieringa, R.: Verification Support for Workflow Design with UML Activity Graphs. In: Proc. ICSE 2002, pp. 166–176 (2002)
9. Gaujal, B., Guia, A.: Optimal Stationary Behavior for a Class of Timed Continuous Petri Nets. Automatica 40, 1505–1516 (2004)
10. Hillston, J.: Fluid Flow Approximation of PEPA Models. In: Proc. 2nd Int. Conf. Quantitative Evaluation Systems, pp. 33–42 (2005)
11. Hiraishi, K.: KCLP-HS: A Rapid Prototyping Tool for Implementing Algorithms on Hybrid Systems, JAIST Research Report IS-RR-2006-012 (August 2006)
12. Horton, G., Kulkarni, V.G., Nicol, D.M., Trivedi, K.S.: Fluid Stochastic Petri Nets: Theory, Applications, and Solution Techniques, NASA Contractor Report, no. 198274 (1996)
13. Júlvez, J., Recald, L., Silva, M.: Steady-state Performance Evaluation of Continuous Mono-t-semiflow Petri Nets. Automatica 41, 605–616 (2005)
14. Kleinrock, L.: Queuing Systems, Volume II: Computer Applications. vol. 2. Wiley, Chichester (1976)
15. Li, J., Fan, Y.S., Zhou, M.C.: Performance Modeling and Analysis of Workflow. IEEE Trans. on Systems, Man, and Cybernetics: Part A 34(2), 229–242 (2004)
16. Lindemann, C.: Performance Modeling with Deterministic and Stochastic Petri Nets. Wiley, Chichester (1998)
17. Mandelbaum, A., Chen, H.: Discrete Flow Networks: Bottlenecks Analysis and Fluid Approximations. Mathematics of Operations Research 16, 408–446 (1991)
18. Moore, R.: Methods and Applications of Interval Analysis. Philadelphia: SIAM (1979)
19. Rauh, A., Kletting, M., Aschemann, H., Hofer, E.P.: Interval Methods for Simulation of Dynamical Systems with State-Dependent Switching Characteristics. In: Proc. IEEE Int. Conf. Control Applications, pp. 355–360 (2006)

20. Silva, M., Recade, L.: Continuization of Timed Petri Nets: From Performance Evaluation to Observation and Control. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 26–47. Springer, Heidelberg (2005)
21. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
22. <http://www.bpmn.org/>
23. <http://rvs.informatik.uni-leipzig.de/de/software/index.php>

Modelling Concurrency with Quotient Monoids

Ryszard Janicki* and Dai Tri Man Lê**

Department of Computing and Software,
McMaster University,
Hamilton, ON, L8S 4K1 Canada
`{janicki,ledt}@mcmaster.ca`

Abstract. Four quotient monoids over step sequences and one with compound generators are introduced and discussed. They all can be regarded as extensions (of various degrees) of Mazurkiewicz traces [14] and comtraces of [10].

Keywords: quotient monoids, traces, comtraces, step sequences, stratified partial orders, stratified order structures, canonical representations.

1 Introduction

Mazurkiewicz traces or partially commutative monoids [1,5] are quotient monoids over sequences (or words). They have been used to model various aspects of concurrency theory since the late seventies and their theory is substantially developed [5]. As a language representation of partial orders, they can nicely model “true concurrency.”

For Mazurkiewicz traces, the basic monoid (whose elements are used in the equations that define the trace congruence) is just a free monoid of sequences. It is assumed that generators, i.e. elements of trace alphabet, have no visible internal structure, so they could be interpreted as just names, symbols, letters, etc. This can be a limitation, as when the generators have some internal structure, for example if they are sets, this internal structure may be used when defining the set of equations that generate the quotient monoid. In this paper we will assume that the monoid generators have some internal structure. We refer to such generators as ‘compound’, and we will use the properties of that internal structure to define an appropriate quotient congruence.

One of the limitations of traces and the partial orders they generate is that neither traces nor partial orders can model the “not later than” relationship [9]. If an event a is performed “not later than” an event b , and let the step $\{a, b\}$ model the simultaneous performance of a and b , then this “not later than” relationship can be modelled by the following set of two step sequences $s = \{\{a\}\{b\}, \{a, b\}\}$. But the set s cannot be represented by any trace. The problem is that the trace independency relation is symmetric, while the “not later than” relationship is not, in general, symmetric.

* Partially supported by NSERC grant of Canada.

** Partially supported by Ontario Graduate Scholarship.

To overcome those limitations the concept of a *comtrace* (*combined trace*) was introduced in [10]. Comtraces are finite sets of equivalent step sequences and the congruence is determined by a relation *ser*, which is called *serialisability* and is, in general, not symmetric. Monoid generators are ‘steps’, i.e. finite sets, so they have internal structure. The set union is used to define comtrace congruence. Comtraces provide a formal language counterpart to *stratified order structures* and were used to provide a semantic of Petri nets with inhibitor arcs. However [10] contains very little theory of comtraces, and only their relationship to stratified order structures has been substantially developed.

Stratified order structures [6,8,10,11] are triples $(X, \prec, \sqsubseteq)$, where $\prec$ and $\sqsubseteq$ are binary relations on X . They were invented to model both “earlier than” (the relation $\prec$) and “not later than” (the relation $\sqsubseteq$) relationships, under the assumption that all system runs are modelled by stratified partial orders, i.e. step sequences. They have been successfully applied to model inhibitor and priority systems, asynchronous races, synthesis problems, etc. (see for example [10,12,13] and others). It was shown in [10] that each comtrace defines a finite stratified order structure. However, thus far, comtraces have been used much less often than stratified order structures, even though in many cases they appear to be more natural than stratified order structures. Perhaps this is due to the lack of substantial theory development of quotient monoids different from that of Mazurkiewicz traces.

It appears that comtraces are a special case of a more general class of quotient monoids, which will be called *absorbing monoids*. For absorbing monoids, generators are still steps, i.e. sets. When sets are replaced by arbitrary compound generators (together with appropriate rules for the generating equations), a new model, called *absorbing monoids with compound generators*, is created. This model allows us to describe formally *asymmetric synchrony*.

Both comtraces and stratified order structures can adequately model concurrent histories only when the paradigm π_3 of [9,11] is satisfied. For the general case, we need *generalised stratified order structures*, which were introduced and analysed in [7]. Generalised stratified order structures are triples $(X, \diamond, \sqsubseteq)$, where $\diamond$ and $\sqsubseteq$ are binary relations on X modelling “earlier than or later than” and “not later than” relationships respectively under the assumption that all system runs are modelled by stratified partial orders.

In this paper a sequence counterpart of generalised stratified order structures, called *generalised comtraces*, and their equational generalisation, called *partially commutative absorbing monoids*, are introduced and their properties are discussed.

In the next section we recall the basic concepts of partial orders and the theory of monoids. Section 3 introduces *equational monoids with compound generators* and other types of monoids that are discussed in this paper. In Section 4 the concept of canonical representations of traces is reviewed; while Section 5 proves the *uniqueness of canonical representations for comtraces*. In Section 6 the notion of *generalised comtraces* is introduced and the relationship between comtraces, generalised comtraces and their respective order structures is

thoroughly discussed. Section 7 briefly describes the relationship between com-traces and different paradigms of concurrent histories, and Section 8 contains some final comments.

2 Orders, Monoids, Sequences and Step Sequences

Let X be a set. A relation $\prec \subseteq X \times X$ is a (*strict*) *partial order* if it is irreflexive and transitive, i.e. if $\neg(a \prec a)$ and $a \prec b \prec c \Rightarrow a \prec c$, for all $a, b, c \in X$.

We write $a \simeq_{\prec} b$ if $\neg(a \prec b) \wedge \neg(b \prec a)$, that is if a and b are either *distinct incomparable* (w.r.t. $\prec$) or *identical* elements of X ; and $a \frown_{\prec} b$ if $a \simeq_{\prec} b \wedge a \neq b$.

We will also write $a \prec\!\!\prec b$ if $a \prec b \vee a \frown_{\prec} b$.

The partial order $\prec$ is *total* (or *linear*) if $\frown_{\prec}$ is empty, and *stratified* (or *weak*) if $\simeq_{\prec}$ is an equivalence relation.

The partial order $\prec_2$ is an *extension* of $\prec_1$ iff $\prec_1 \subseteq \prec_2$. Every partial order is uniquely represented by the intersection of all its total extensions.

A triple $(X, \circ, 1)$, where X is a set, $\circ$ is a total binary operation on X , and $1 \in X$, is called a *monoid*, if $(a \circ b) \circ c = a \circ (b \circ c)$ and $a \circ 1 = 1 \circ a = a$, for all $a, b, c \in X$.

A relation $\sim \subseteq X \times X$ is a *congruence* in the monoid $(X, \circ, 1)$ if

$$a_1 \sim b_1 \wedge a_2 \sim b_2 \Rightarrow (a_1 \circ a_2) \sim (b_1 \circ b_2),$$

for all $a_1, a_2, b_1, b_2 \in X$. Every congruence is an equivalence relation; standardly $X/\sim$ denotes the set of all equivalence classes of $\sim$ and $[a]_{\sim}$ (or simply $[a]$) denotes the equivalence class of $\sim$ containing the element $a \in X$. The triple $(X/\sim, \hat{\circ}, [1])$, where $[a]\hat{\circ}[b] = [a \circ b]$, is called the *quotient monoid* of $(X, \circ, 1)$ under the congruence $\sim$. The mapping $\phi : X \rightarrow X/\sim$ defined as $\phi(a) = [a]$ is called the *natural homomorphism* generated by the congruence $\sim$ (for more details see for example [2]). The symbols $\circ$ and $\hat{\circ}$ are often omitted if this does not lead to any discrepancy.

By an *alphabet* we shall understand any finite set. For an alphabet Σ , Σ^* denotes the set of all finite sequences of elements of Σ , λ denotes the empty sequence, and any subset of Σ^* is called a *language*. In this paper all sequences are finite. Each sequence can be interpreted as a total order and each finite total order can be represented by a sequence. The triple $(\Sigma^*, \cdot, \lambda)$, where $\cdot$ is sequence concatenation (usually omitted), is a *monoid* (of sequences).

For each set X , let $\mathcal{P}(X)$ denote the set of all subsets of X and $\mathcal{P}^{\emptyset}(X)$ denote the set of all *non-empty* subsets of X . Consider an alphabet $\Sigma_{step} \subseteq \mathcal{P}^{\emptyset}(X)$ for some finite X . The elements of Σ_{step} are called *steps* and the elements of Σ_{step}^* are called *step sequences*. For example if $\Sigma_{step} = \{\{a\}, \{a, b\}, \{c\}, \{a, b, c\}\}$ then $\{a, b\}\{c\}\{a, b, c\} \in \Sigma_{step}^*$ is a step sequence. The triple $(\Sigma_{step}^*, \bullet, \lambda)$, where $\bullet$ is step sequence concatenation (usually omitted), is a *monoid* (of step sequences) (see for example [10] for details).

3 Equational Monoids with Compound Generators

In this section we will define all types of monoids that are discussed in this paper.

3.1 Equational Monoids and Mazurkiewicz Traces

Let $M = (X, \circ, 1)$ be a *monoid* and let $EQ = \{x_1 = y_1, \dots, x_n = y_n\}$, where $x_i, y_i \in X$, $i = 1, \dots, n$, be a finite set of *equations*. Define $\equiv_{EQ}$ (or just $\equiv$) as the *least congruence* on M satisfying, $x_i = y_i \implies x_i \equiv_{EQ} y_i$, for each equation $x_i = y_i \in EQ$. We will call the relation $\equiv_{EQ}$ the *congruence defined by EQ*, or *EQ-congruence*.

The *quotient monoid* $M_{\equiv} = (X/\equiv, \hat{\circ}, [1]_{\equiv})$, where $[x]\hat{\circ}[y] = [x \circ y]$, will be called an *equational monoid* (see for example [15]).

The following folklore result shows that the relation $\equiv_{EQ}$ can also be defined explicitly.

Proposition 1. *For equational monoids the EQ-congruence $\equiv$ can be defined explicitly as the reflexive and transitive closure of the relation $\approx \cup \approx^{-1}$, i.e. $\equiv = (\approx \cup \approx^{-1})^*$, where $\approx \subseteq X \times X$, and*

$$x \approx y \iff \exists x_1, x_2 \in X. \exists (u = w) \in EQ. x = x_1 \circ u \circ x_2 \wedge y = x_1 \circ v \circ x_2.$$

Proof. Define $\dot{\approx} = \approx \cup \approx^{-1}$. Clearly $(\dot{\approx})^*$ is an equivalence relation. Let $x_1 \equiv y_1$ and $x_2 \equiv y_2$. This means $x_1(\dot{\approx})^k y_1$ and $x_2(\dot{\approx})^l y_2$ for some $k, l \geq 0$. Hence $x_1 \circ x_2 (\dot{\approx})^k y_1 \circ x_2 (\dot{\approx})^l y_1 \circ y_2$, i.e. $x_1 \circ x_2 \equiv y_1 \circ y_2$. Therefore $\equiv$ is a congruence. Let $\sim$ be a congruence that satisfies $(u = w) \in EQ \implies u \sim w$ for each $u = w$ from EQ . Clearly $x \dot{\approx} y \implies x \sim y$. Hence $x \equiv y \iff x(\dot{\approx})^m y \implies x \sim^m y \Rightarrow x \sim y$. Thus $\equiv$ is the least. $\square$

If $M = (E^*, \circ, \lambda)$ is a *free monoid* generated by E , $ind \subseteq E \times E$ is an irreflexive and symmetric relation (called *independency* or *commutation*), and $EQ = \{ab = ba \mid (a, b) \in ind\}$, then the quotient monoid $M_{\equiv} = (E^*/\equiv, \hat{\circ}, [\lambda])$ is a *partially commutative free monoid* or *monoid of Mazurkiewicz traces* [5,14]. The tuple (E, ind) is often called *concurrent alphabet*.

Example 1. Let $E = \{a, b, c\}$, $ind = \{(b, c), (c, b)\}$, i.e. $EQ = \{bc = cb\}$. For example $abcbca \equiv acbbca$ (since $abcbca \approx acbbca \approx acbcbca \approx accbba$), $t_1 = [abc] = \{abc, acb\}$, $t_2 = [bca] = \{bca, cba\}$ and $t_3 = [abcbca] = \{abcbca, abccba, acbbca, acbcbca, abbcca, accbba\}$ are traces, and $t_3 = t_1 \hat{\circ} t_2$ (as $[abcbca] = [abc] \hat{\circ} [bca]$). For more details the reader is referred to [5,14] (and [15] for equational representations). $\square$

3.2 Absorbing Monoids and Comtraces

The standard definition of a free monoid $(E^*, \circ, \lambda)$ assumes the elements of E have no internal structure (or their internal structure does not affect any monoidal properties), and they are often called ‘letters’, ‘symbols’, ‘names’, etc. When we assume the elements of E have some internal structure, for instance they are sets, this internal structure may be used when defining the set of equations EQ .

Let E be a finite set and $\mathcal{S} \subseteq \mathcal{P}^{\emptyset}(E)$ be a non-empty set of non-empty subsets of E satisfying $\bigcup_{A \in \mathcal{S}} A = E$. The free monoid $(\mathcal{S}^*, \circ, \lambda)$ is called a *free monoid*

of step sequences over E , with the elements of $\mathcal{S}$ called *steps* and the elements of $\mathcal{S}^*$ called *step sequences*. We assume additionally that the set $\mathcal{S}$ is *subset closed* i.e. for all $A \in \mathcal{S}$, $B \subseteq A$ and B is not empty, implies $B \in \mathcal{S}$.

Let EQ be the following set of equations:

$$EQ = \{ C_1 = A_1 B_1, \dots, C_n = A_n B_n \},$$

where $A_i, B_i, C_i \in \mathcal{S}$, $C_i = A_i \cup B_i$, $A_i \cap B_i = \emptyset$, for $i = 1, \dots, n$, and let $\equiv$ be EQ -congruence (i.e. the least congruence satisfying $C_i = A_i B_i$ implies $C_i \equiv A_i B_i$).

The quotient monoid $(\mathcal{S}^*/\equiv, \hat{\circ}, [\lambda])$ will be called an **absorbing monoid over step sequences**.

Example 2. Let $E = \{a, b, c\}$, $\mathcal{S} = \{\{a, b, c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}\}$, and EQ be the following set of equations:

$$\{a, b, c\} = \{a, b\}\{c\} \quad \text{and} \quad \{a, b, c\} = \{a\}\{b, c\}.$$

In this case, for example, $\{a, b\}\{c\}\{a\}\{b, c\} \equiv \{a\}\{b, c\}\{a, b\}\{c\}$ (as we have $\{a, b\}\{c\}\{a\}\{b, c\} \approx \{a, b, c\}\{a\}\{b, c\} \approx \{a, b, c\}\{a, b, c\} \approx \{a\}\{b, c\}\{a, b, c\} \approx \{a\}\{b, c\}\{a, b\}\{c\}$), $x = [\{a, b, c\}]$ and $y = [\{a, b\}\{c\}\{a\}\{b, c\}]$ belong to $\mathcal{S}^*/\equiv$, and

$$\begin{aligned} x &= \{\{a, b, c\}, \{a, b\}\{c\}, \{a\}\{b, c\}\}, \\ y &= \{\{a, b, c\}\{a, b, c\}, \{a, b, c\}\{a, b\}\{c\}, \{a, b, c\}\{a\}\{b, c\}, \{a, b\}\{c\}\{a, b, c\}, \\ &\quad \{a, b\}\{c\}\{a, b\}\{c\}, \{a, b\}\{c\}\{a\}\{b, c\}, \{a\}\{b, c\}\{a, b, c\}, \\ &\quad \{a\}\{b, c\}\{a, b\}\{c\}, \{a\}\{b, c\}\{a\}\{b, c\}\}. \end{aligned}$$

Note that $y = x \hat{\circ} x$ as $\{a, b\}\{c\}\{a\}\{b, c\} \equiv \{a, b, c\}\{a, b, c\}$. □

Comtraces, introduced in [10] as an extension of Mazurkiewicz traces to distinguish between “earlier than” and “not later than” phenomena, are a special case of absorbing monoids of step sequences. The equations EQ are in this case defined implicitly via two relations *simultaneity* and *serialisability*.

Let E be a finite set (of events), $ser \subseteq sim \subseteq E \times E$ be two relations called *serialisability* and *simultaneity* respectively. The triple (E, sim, ser) is called *comtrace alphabet*. We assume that *sim* is irreflexive and symmetric. Intuitively, if $(a, b) \in sim$ then a and b can occur simultaneously (or be a part of a *synchronous* occurrence in the sense of [12]), while $(a, b) \in ser$ means that a and b may occur simultaneously and a may occur before b (and both happenings are equivalent). We define $\mathcal{S}$, the set of all (potential) *steps*, as the set of all cliques of the graph (E, sim) , i.e.

$$\mathcal{S} = \{A \mid A \neq \emptyset \wedge (\forall a, b \in A. a = b \vee (a, b) \in sim)\}.$$

The set of equations EQ can now be defined as:

$$EQ = \{C = AB \mid C = A \cup B \in \mathcal{S} \wedge A \cap B = \emptyset \wedge A \times B \subseteq ser\}.$$

Let $\equiv$ be the EQ -congruence defined by the above set of equations. The absorbing monoid $(\mathcal{S}/\equiv, \hat{\circ}, [\lambda])$ is called a monoid of *comtraces*.

Example 3. Let $E = \{a, b, c\}$ where a, b and c are three atomic operations defined as follows (we assume simultaneous reading is allowed):

$$a : y \leftarrow x + y, \quad b : x \leftarrow y + 2, \quad c : y \leftarrow y + 1.$$

Only b and c can be performed simultaneously, and the simultaneous execution of b and c gives the same outcome as executing b followed by c . We can then define $sim = \{(b, c), (c, b)\}$ and $ser = \{(b, c)\}$, and we have $\mathcal{S} = \{\{a\}, \{b\}, \{c\}, \{b, c\}\}$, $EQ = \{\{b, c\} = \{b\}\{c\}\}$. For example $x = [\{a\}\{b, c\}] = \{\{a\}\{b, c\}, \{a\}\{b\}\{c\}\}$ is a comtrace. Note that $\{a\}\{c\}\{b\} \notin x$. $\square$

Even though Mazurkiewicz traces are quotient monoids over sequences and comtraces are quotient monoids over step sequences, Mazurkiewicz traces can be regarded as a special case of comtraces. In principle, each trace commutativity equation $ab = ba$ corresponds to two comtrace absorbing equations $\{a, b\} = \{a\}\{b\}$ and $\{a, b\} = \{b\}\{a\}$. This relationship can formally be formulated as follows.

Proposition 2. *If $ser = sim$ then for each comtrace $t \in \mathcal{S}^* / \equiv_{ser}$ there is a step sequence $x = \{a_1\} \dots \{a_k\} \in \mathcal{S}^*$, where $a_i \in E$, $i = 1, \dots, k$ such that $t = [x]$.*

Proof. Let $t = [A_1 \dots A_m]$, where $A_i \in \mathcal{S}$, $i = 1, \dots, m$. Hence $t = [A_1] \dots [A_m]$. Let $A_i = \{a_1^i, \dots, a_{k_i}^i\}$. Since $ser = sim$, we have $[A_i] = [\{a_1^i\}] \dots [\{a_{k_i}^i\}]$, for $i = 1, \dots, m$, which ends the proof. $\square$

This means that if $ser = sim$, then each comtrace $t \in \mathcal{S}^* / \equiv_{ser}$ can be represented by a Mazurkiewicz trace $[a_1 \dots a_k] \in E^* / \equiv_{ind}$, where $ind = ser$ and $\{a_1\} \dots \{a_k\}$ is a step sequence such that $t = [\{a_1\} \dots \{a_k\}]$. Proposition 2 guarantees the existence of $a_1 \dots a_k$.

While every comtrace monoid is an absorbing monoid, *not every* absorbing monoid can be defined as a comtrace. For example the absorbing monoid analysed in Example 2 *cannot* be represented by any comtrace monoid.

It appears the concept of the comtrace can be very useful to formally define the concept of *synchrony* (in the sense of [12]). In principle the events are *synchronous* if they *can* be executed in one step $\{a_1, \dots, a_k\}$ but this execution *cannot* be modelled by any sequence of proper subsets of $\{a_1, \dots, a_k\}$. In general ‘synchrony’ is not necessarily ‘simultaneity’ as it does not include the concept of time [4]. However, it appears that the mathematics used to deal with synchrony is very close to that to deal with simultaneity.

Let (E, sim, ser) be a given comtrace alphabet. We define the relations ind , syn and the set $\mathcal{S}_{syn}$ as follows:

- $ind \subseteq E \times E$, called *independency* and defined as $ind = ser \cap ser^{-1}$,
- $syn \subseteq E \times E$, called *synchrony* and defined as:
 $(a, b) \in syn \iff (a, b) \in sim \wedge (a, b) \notin ser \cup ser^{-1}$,
- $\mathcal{S}_{syn} \subseteq \mathcal{S}$, called *synchronous steps*, and defined as:
 $A \in \mathcal{S}_{syn} \iff A \neq \emptyset \wedge (\forall a, b \in A. (a, b) \in syn)$.

If $(a, b) \in ind$ then a and b are *independent*, i.e. they may be executed either simultaneously, or a followed by b , or b followed by a , with exactly the same result. If $(a, b) \in syn$ then a and b are *synchronous*, which means they might be executed in one step, either $\{a, b\}$ or as a part of bigger step, but such an execution is not equivalent to neither a followed by b , nor b followed by a . In

principle, the relation syn is a counterpart of ‘synchrony’ as understood in [12]. If $A \in \mathcal{S}_{syn}$ then the set of events A can be executed as one step, but it *cannot* be simulated by any sequence of its subsets.

Example 4. Let $E = \{a, b, c, d, e\}$, $sim = \{(a, b), (b, a), (a, c), (c, a), (a, d), (d, a)\}$, and $ser = \{(a, b), (b, a), (a, c)\}$. Hence

$$\mathcal{S} = \{\{a, b\}, \{a, c\}, \{a, d\}, \{a\}, \{b\}, \{c\}, \{e\}\}, ind = \{(a, b), (b, a)\},$$

$$syn = \{(a, d), (d, a)\}, \mathcal{S}_{syn} = \{\{a, d\}\}.$$

Since $\{a, d\} \in \mathcal{S}_{syn}$ the step $\{a, d\}$ *cannot* be split. For example the comtraces $x_1 = [\{a, b\}\{c\}\{a\}]$, $x_2 = [\{e\}\{a, d\}\{a, c\}]$, $x_3 = [\{a, b\}\{c\}\{a\}\{e\}\{a, d\}\{a, c\}]$, are the following sets of step sequences:

$$\begin{aligned} x_1 &= \{\{a, b\}\{c\}\{a\}, \{a\}\{b\}\{c\}\{a\}, \{b\}\{a\}\{c\}\{a\}, \{b\}\{a, c\}\{a\}\}, \\ x_2 &= \{\{e\}\{a, d\}\{a, c\}, \{e\}\{a, d\}\{a\}\{c\}\}, \\ x_3 &= \{\{a, b\}\{c\}\{a\}\{e\}\{a, d\}\{a, c\}, \{a\}\{b\}\{c\}\{a\}\{e\}\{a, d\}\{a, c\}, \\ &\quad \{b\}\{a\}\{c\}\{a\}\{e\}\{a, d\}\{a, c\}, \{b\}\{a, c\}\{a\}\{e\}\{a, d\}\{a, c\}, \\ &\quad \{a, b\}\{c\}\{a\}\{e\}\{a, d\}\{a\}\{c\}, \{a\}\{b\}\{c\}\{a\}\{e\}\{a, d\}\{a\}\{c\}, \\ &\quad \{b\}\{a\}\{c\}\{a\}\{e\}\{a, d\}\{a\}\{c\}, \{b\}\{a, c\}\{a\}\{e\}\{a, d\}\{a\}\{c\}\}. \end{aligned}$$

Notice that we have $\{a, c\} \equiv_{ser} \{a\}\{c\} \not\equiv_{ser} \{c\}\{a\}$, since $(c, a) \notin ser$. We also have $x_3 = x_1 \hat{\circ} x_2$. $\square$

3.3 Partially Commutative Absorbing Monoids and Generalised Comtraces

There are reasonable concurrent histories that cannot be modelled by any absorbing monoid. In fact, absorbing monoids can only model concurrent histories conforming to the paradigm π_3 of [9] (see the Section 7 of this paper). Let us analyse the following example.

Example 5. Let $E = \{a, b, c\}$ where a, b and c are three atomic operations defined as follows (we assume simultaneous reading is allowed):

$$a : x \leftarrow x + 1, \quad b : x \leftarrow x + 2, \quad c : y \leftarrow y + 1.$$

It is reasonable to consider them all as ‘concurrent’ as any order of their executions yields exactly the same results (see [9,11] for more motivation and formal considerations). Note that while simultaneous execution of $\{a, c\}$ and $\{b, c\}$ are allowed, the step $\{a, b\}$ *is not!*

Let us consider set of all equivalent executions (or runs) involving one occurrence of a, b and c

$$\begin{aligned} x = & \{\{a\}\{b\}\{c\}, \{a\}\{c\}\{b\}, \{b\}\{a\}\{c\}, \{b\}\{c\}\{a\}, \{c\}\{a\}\{b\}, \{c\}\{b\}\{a\}, \\ & \{a, c\}\{b\}, \{b, c\}\{a\}, \{b\}\{a, c\}, \{a\}\{b, c\}\}. \end{aligned}$$

Although x is a valid concurrent history or behaviour [9,11], it *is not* a comtrace. The problem is that we have here $\{a\}\{b\} = \{b\}\{a\}$ but $\{a, b\}$ *is not* a valid step, so no absorbing monoid can represent this situation. $\square$

The concurrent behaviour described by x from Example 5 can easily be modelled by a *generalised order structure* of [7]. In this subsection we will introduce the concept of *generalised comtraces*, quotient monoids representations of generalised

order structures. But we start with a slightly more general concept of *partially commutative absorbing monoid* over step sequences.

Let E be a finite set and let $(\mathcal{S}^*, \circ, \lambda)$ be a free monoid of step sequences over E . Assume also that $\mathcal{S}$ is subset closed.

Let EQ, EQ', EQ'' be the following sets of equations:

$$EQ' = \{ C'_1 = A'_1 B'_1, \dots, C'_n = A'_n B'_n \},$$

where $A'_i, B'_i, C'_i \in \mathcal{S}$, $C'_i = A'_i \cup B'_i$, $A'_i \cap B'_i = \emptyset$, for $i = 1, \dots, n$,

$$EQ'' = \{ B''_1 A''_1 = A''_1 B''_1, \dots, B''_k A''_k = A''_k B''_k \},$$

where $A''_i, B''_i \in \mathcal{S}$, $A''_i \cap B''_i = \emptyset$, $A''_i \cup B''_i \notin \mathcal{S}$, for $i = 1, \dots, k$, and

$$EQ = EQ' \cup EQ''.$$

Let $\equiv$ be the EQ -congruence defined by the above set of equations EQ (i.e. the least congruence such that $C'_i = A'_i B'_i \implies C'_i \equiv A'_i B'_i$, for $i = 1, \dots, n$ and $B''_i A''_i = A''_i B''_i \implies B''_i A''_i \equiv A''_i B''_i$, for $i = 1, \dots, k$). The quotient monoid $(\mathcal{S}/\equiv, \hat{\circ}, [\lambda])$ will be called a ***partially commutative absorbing monoid over step sequences***.

There is a *substantial difference* between $ab = ba$ for Mazurkiewicz traces, and $\{a\}\{b\} = \{b\}\{a\}$ for partially commutative absorbing monoids. For traces, the equation $ab = ba$ when translated into step sequences corresponds to $\{a, b\} = \{a\}\{b\}$, $\{a, b\} = \{b\}\{a\}$, and implies $\{a\}\{b\} \equiv \{b\}\{a\}$. For partially commutative absorbing monoids, the equation $\{a\}\{b\} = \{b\}\{a\}$ implies that $\{a, b\}$ is not a step, i.e. neither $\{a, b\} = \{a\}\{b\}$ nor $\{a, b\} = \{b\}\{a\}$ does exist. For Mazurkiewicz traces the equation $ab = ba$ means ‘independency’, i.e. any order or simultaneous execution are allowed and are equivalent. For partially commutative absorbing monoids, the equation $\{a\}\{b\} = \{b\}\{a\}$ means that both orders are equivalent but simultaneous execution does not exist.

We will now extend the concept of a comtrace by adding a relation that generates the set of equations EQ'' .

Let E be a finite set (of events), $ser, sim, inl \subset E \times E$ be three relations called *serialisability*, *simultaneity* and *interleaving* respectively. The triple (E, sim, ser, inl) is called *generalised comtrace alphabet*. We assume that both sim and inl are irreflexive and symmetric, and

$$ser \subseteq sim, \quad sim \cap inl = \emptyset.$$

Intuitively, if $(a, b) \in sim$ then a and b can occur simultaneously (or be a part of a *synchronous* occurrence), $(a, b) \in ser$ means that a and b may occur simultaneously and a may occur before b (and both happenings are equivalent), and $(a, b) \in inl$ means a and b cannot occur simultaneously, but their occurrence in any order is equivalent. As for comtraces, we define $\mathcal{S}$, the set of all (potential) *steps*, as the set of all cliques of the graph (E, sim) .

The set of equations EQ can now be defined as $EQ = EQ' \cup EQ''$, where:

$$EQ' = \{ C = AB \mid C = A \cup B \in \mathcal{S} \wedge A \cap B = \emptyset \wedge A \times B \subseteq ser \}, \text{ and}$$

$$EQ'' = \{ BA = AB \mid A \cup B \notin \mathcal{S} \wedge A \cap B = \emptyset \wedge A \times B \subseteq inl \}.$$

Let $\equiv$ be the EQ -congruence defined by the above set of equations. The quotient monoid $(\mathcal{S}^*/\equiv, \hat{\circ}, [\lambda])$ is called a monoid of **generalised comtraces**. If inl is empty we have a monoid of comtraces.

Example 6. The set x from Example 5 is an element of the generalised comtrace with $E = \{a, b, c\}$, $ser = sim = \{(a, c), (c, a), (b, c), (c, a)\}$, $inl = \{(a, b), (b, a)\}$, and $\mathcal{S} = \{\{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}\}$, and for example $x = [\{a, c\}\{b\}]$. $\square$

3.4 Absorbing Monoids with Compound Generators

One of the concepts that *cannot* easily be modelled by quotient monoids over step sequences, is *asymmetric synchrony*. Consider the following example.

Example 7. Let a and b be atomic and potentially simultaneous events, and c_1 , c_2 be two synchronous compound events built entirely from a and b . Assume that c_1 is equivalent to the sequence $a \circ b$, c_2 is equivalent to the sequence $b \circ a$, but c_1 is *not* equivalent to c_2 . This situation cannot be modelled by steps as from a and b we can built only one step $\{a, b\} = \{b, a\}$. To provide more intuition consider the following simple problem.

Assume we have a buffer of 8 bits. Each event a and b consecutively fills 4 bits. The buffer is initially empty and whoever starts first fills the bits 1–4 and whoever starts second fills the bits 5–8. Suppose that the simultaneous start is impossible (begins and ends are instantaneous events after all), filling the buffer takes time, and simultaneous (i.e. time overlapping in this case) executions are allowed. We clearly have two synchronous events $c_1 = \text{'}a \text{ starts first but overlaps with } b\text{'}$, and $c_2 = \text{'}b \text{ starts first but overlaps with } a\text{'}$. We now have $c_1 = a \circ b$, and $c_2 = b \circ a$, but obviously $c_1 \neq c_2$ and $c_1 \not\equiv c_2$. $\square$

To adequately model situations like that in Example 7 we will introduce the concept of *absorbing monoid with compound generators*.

Let $(G^*, \circ, \lambda)$ be a free monoid generated by G , where $G = E \cup C$, $E \cap C = \emptyset$. The set E is the set of *elementary* generators, while the set C is the set of *compound* generators. We will call $(G^*, \circ, \lambda)$ a *free monoid with compound generators*.

Assume we have a function $comp : G \rightarrow \mathcal{P}^\emptyset(E)$, called *composition* that satisfies: for all $a \in E$, $comp(a) = \{a\}$ and for all $a \notin E$, $|comp(a)| \geq 2$.

For each $a \in G$, $comp(a)$ gives the set of all elementary elements from which a is composed. It *may happen* that $comp(a) = comp(b)$ and $a \neq b$.

The set of absorbing equations is defined as follows:

$$EQ = \{c_i = a_i \circ b_i \mid i = 1, \dots, n\}$$

where for each $i = 1, \dots, n$, we have:

- $a_i, b_i, c_i \in G$,
- $comp(c_i) = comp(a_i) \cup comp(b_i)$,
- $comp(a_i) \cap comp(b_i) = \emptyset$.

Let $\equiv$ be the EQ -congruence defined by the above set of equations EQ . The quotient monoid $(G^*/\equiv, \hat{\circ}, [\lambda])$ is called an **absorbing monoid with compound generators**.

Example 8. Consider the absorbing monoid with compound generators where: $E = \{a, b, c_1, c_2\}$, $comp(c_1) = comp(c_2) = \{a, b\}$, $comp(a) = \{a\}$, $comp(b) = \{b\}$, and $EQ = \{ c_1 = a \circ b, \quad c_2 = b \circ a \}$. Now we have $[c_1] = \{c_1, a \circ b\}$ and $[c_2] = \{c_2, b \circ a\}$, which models the case from Example 7. $\square$

4 Canonical Representations

We will show that all of the kinds of monoids discussed in previous sections have some kind of *canonical* representation, which intuitively corresponds to a maximally concurrent execution of concurrent histories [3].

Let (E, ind) be a concurrent alphabet and $(E^*/\equiv, \hat{\circ}, [\lambda])$ be a monoid of Mazurkiewicz traces. A sequence $x = a_1 \dots a_k \in E^*$ is called *fully commutative* if $(a_i, a_j) \in ind$ for all $i \neq j$ and $i, j = 1, \dots, k$.

A sequence $x \in E^*$ is in the *canonical form* if $x = \lambda$ or $x = x_1 \dots x_n$ such that

- each x_i is fully commutative, for $i = 1, \dots, n$,
- for each $1 \leq i \leq n-1$ and for each element a of x_{i+1} there exists an element b of x_i such that $a \neq b$ and $(a, b) \notin ind$.

If x is in the canonical form, then x is a *canonical representation* of $[x]$.

Theorem 1 ([1,3]). *For every trace $t \in E^*/\equiv$, there exists $x \in E^*$ such that $t = [x]$ and x is in the canonical form.* $\square$

With the canonical form as defined above, a trace may have more than one canonical representations. For instance the trace $t_3 = [abcbca]$ from Example 1 has four canonical representations: $abcbca, acbbca, abccba, acbcb a$. Intuitively, x in the canonical form represents the maximally concurrent execution of a concurrent history represented by $[x]$. In this representation fully commutative sequences built from the same elements can be considered equivalent (this is better seen when *vector firing sequences* of [16] are used to represent traces, see [3] for more details). To get the uniqueness it suffices to order fully commutative sequences. For example we may introduce an arbitrary total order on E , extend it lexicographically to E^* and add the condition that in the representation $x = x_1 \dots x_n$, each x_i is minimal with the lexicographic ordering. The canonical form with this additional condition is called *Foata canonical form*.

Theorem 2 ([1]). *Every trace has a unique representation in the Foata canonical form.* $\square$

A canonical form as defined at the beginning of this Section can easily be adapted to step sequences and various equational monoids over step sequences, as well as to monoids with compound generators. In fact, step sequences better represent the intuition that canonical representation corresponds to the maximally concurrent execution [3].

Let $(\mathcal{S}^*, \circ, \lambda)$ be a free monoid of step sequences over E , and let

$$EQ = \{ C_1 = A_1 B_1, \dots, C_n = A_n B_n \}$$

be an appropriate set of absorbing equations. Let $M_{absorb} = (\mathcal{S}^*/\equiv, \hat{\circ}, [\lambda])$.

A step sequence $t = A_1 \dots A_k \in \mathcal{S}^*$ is *canonical* (w.r.t. M_{absorb}) if for all $i \geq 2$ there is *no* $B \in \mathcal{S}$ satisfying:

$$\begin{aligned} (A_{i-1} \cup B = A_{i-1}B) &\in EQ \\ (A_i = B(A_i - B)) &\in EQ \end{aligned}$$

When M_{absorb} is a monoid of comtraces, the above definition is equivalent to the definition of canonical step sequence proposed in [10].

Let $(\mathcal{S}^*, \circ, \lambda)$ be a free monoid of step sequences over E , and let

$$\begin{aligned} EQ' &= \{ C'_1 = A'_1 B'_1, \dots, C'_n = A'_n B'_n \}, \\ EQ'' &= \{ B''_1 A''_1 = A''_1 B''_1, \dots, B''_k A''_k = A''_k B''_k \} \end{aligned}$$

be an appropriate set of partially commutative absorbing equations. Then let $M_{abs\&pc} = (\mathcal{S}^* / \equiv, \hat{\circ}, [\lambda])$.

A step sequence $t = A_1 \dots A_k \in \mathcal{S}^*$ is *canonical* (w.r.t. $M_{abs\&pc}$) if for all $i \geq 2$ there is *no* $B \in \mathcal{S}$ satisfying:

$$\begin{aligned} (A_{i-1} \cup B = A_{i-1}B) &\in EQ' \\ (A_i = B(A_i - B)) &\in EQ' \end{aligned}$$

Note that the set of equations EQ'' *does not* appear in the above definition. Clearly the above definition also applies to generalised comtraces.

Let $(G^*, \circ, \lambda)$ be a free monoid with compound generators, and let

$$EQ = \{ c_1 = a_1 b_1, \dots, c_n = a_n b_n \}$$

be an appropriate set of absorbing equations. Let $M_{cg\&absorb} = (G^* / \equiv, \hat{\circ}, [\lambda])$.

Finally, a sequence $t = a_1 \dots a_k \in G^*$ is *canonical* (w.r.t. $M_{cg\&absorb}$) if for all $i \geq 2$ there is *no* $b, d \in G$ satisfying:

$$\begin{aligned} (c = a_{i-1}b) &\in EQ \\ (a_i = bd) &\in EQ \end{aligned}$$

For all above definitions, if x is in the canonical form, then x is a *canonical representation* of $[x]$.

Theorem 3. *Let M_{absorb} be an absorbing monoid over step sequences, $\mathcal{S}$ its set of steps, and EQ its set of absorbing equations. For every step sequence $t \in \mathcal{S}^*$ there is a canonical step sequence u such that $t \equiv u$.*

Proof. For every step sequence $x = B_1 \dots B_r$, let $\mu(x) = 1 \cdot |B_1| + \dots + r \cdot |B_r|$. There is (at least one) $u \in [t]$ such that $\mu(u) \leq \mu(x)$ for all $x \in [t]$. Suppose $u = A_1 \dots A_k$ is not canonical. Then there is $i \geq 2$ and a step $B \in \mathcal{S}$ satisfying:

$$\begin{aligned} (A_{i-1} \cup B = A_{i-1}B) &\in EQ \\ (A_i = B(A_i - B)) &\in EQ \end{aligned}$$

If $B = A_i$ then $w \approx u$ and $\mu(w) < \mu(u)$, where

$$w = A_1 \dots A_{i-2} (A_{i-1} \cup A_i) A_{i+1} \dots A_k.$$

If $B \neq A_i$, then $w \approx z$ and $u \approx z$ and $\mu(w) < \mu(u)$, where

$$\begin{aligned} z &= A_1 \dots A_{i-2} A_{i-1} B (A_i - B) A_{i+1} \dots A_k \\ w &= A_1 \dots A_{i-2} (A_{i-1} \cup B) (A_i - B) A_{i+1} \dots A_k. \end{aligned}$$

In both cases it contradicts the minimality of $\mu(u)$. Hence u is canonical. $\square$

For partially commutative absorbing monoids over step sequences the proof is virtually identical, the only change is to replace EQ with EQ' . The proof can also be adapted (some ‘notational’ changes only) to absorbing monoids with compound generators.

Corollary 1. *Let $M = (X, \hat{\circ}, [\lambda])$ be an absorbing monoid over step sequences, or partially commutative absorbing monoid over step sequences, or absorbing monoid with compound generators. For every $x \in X$ there is a canonical sequence u such that $x = [u]$.* $\square$

Unless additional properties are assumed, the canonical representation is not unique for all three kinds of monoids mentioned in Corollary 1. To prove this, it suffices to show that this is not true for the absorbing monoids over step sequences. Consider the following example.

Example 9. Let $E = \{a, b, c\}$, $\mathcal{S} = \{\{a, b\}, \{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}\}$ and EQ be the the following set of equations:

$$\{a, b\} = \{a\}\{b\}, \quad \{a, c\} = \{a\}\{c\}, \quad \{b, c\} = \{b\}\{c\}, \quad \{b, c\} = \{c\}\{b\}.$$

Note that $\{a, b\}\{c\}$ and $\{a, c\}\{b\}$ are canonical step sequences, and $\{a, b\}\{c\} \approx \{a\}\{b\}\{c\} \approx \{a\}\{b, c\} \approx \{a\}\{c\}\{b\} \approx \{a, c\}\{b\}$, i.e. $\{a, b\}\{c\} \equiv \{a, c\}\{b\}$. Hence $[\{a, b\}\{c\}] = \{\{a, b\}\{c\}, \{a\}\{b\}\{c\}, \{a\}\{c\}\{b\}, \{a, c\}\{b\}\}$, has two canonical representations $\{a, b\}\{c\}$ and $\{a, c\}\{b\}$. One can easily check that this absorbing monoid is not a monoid of comtraces. $\square$

The canonical representation is also not unique for generalised comtraces if $inl \neq \emptyset$. For any generalised comtrace, if $\{a, b\} \subseteq E$, $(a, b) \in inl$, then $x = [\{a\}\{b\}] = \{\{a\}\{b\}, \{b\}\{a\}\}$ and x has two canonical representations $\{a\}\{b\}$ and $\{b\}\{a\}$.

All the canonical representations discussed above can be extended to unique canonical representations by simply introducing some total order on step sequences, and adding a minimality requirement with respect to this total order to the definition of a canonical form. The technique used in the definition of Foata normal form is one possibility. However this has nothing to do with any property of concurrency and hence will not be discussed in this paper.

However the comtraces have a unique canonical representation as defined above. This was not proved in [10] and will be proved in the next section.

5 Canonical Representations of Comtraces

In principle the uniqueness of canonical representation for comtraces follows the fact that all equations can be derived from the properties of pairs of events. This

results in very strong cancellation and projection properties, and very regular structure of the set of all steps $\mathcal{S}$.

Let $a \in E$ and $w \in \mathcal{S}^*$. We can define a *right cancellation* operator $\div_R$ as

$$\lambda \div_R a = \lambda, \quad wA \div_R a = \begin{cases} (w \div_R a)A & \text{if } a \notin A \\ w & \text{if } A = \{a\} \\ w(A \setminus \{a\}) & \text{otherwise.} \end{cases}$$

Symmetrically, a *left cancellation* operator $\div_L$ is defined as

$$\lambda \div_L a = \lambda, \quad Aw \div_L a = \begin{cases} A(w \div_L a) & \text{if } a \notin A \\ w & \text{if } A = \{a\} \\ (A \setminus \{a\})w & \text{otherwise.} \end{cases}$$

Finally, for each $D \subseteq E$, we define the function $\pi_D : \mathcal{S}^* \rightarrow \mathcal{S}^*$, *step sequence projection* onto D , as follows:

$$\pi_D(\lambda) = \lambda, \quad \pi_D(wA) = \begin{cases} \pi_D(w) & \text{if } A \cap D = \emptyset \\ \pi_D(w)(A \cap D) & \text{otherwise.} \end{cases}$$

Proposition 3.

1. $u \equiv v \implies u \div_R a \equiv v \div_R a$. (right cancellation)
2. $u \equiv v \implies u \div_L a \equiv v \div_L a$. (left cancellation)
3. $u \equiv v \implies \pi_D(u) \equiv \pi_D(v)$. (projection rule)

Proof. For each step sequence $t = A_1 \dots A_k \in \mathcal{S}^*$ let $\Sigma(t) = \bigcup_{i=1}^k A_i$. Note that for comtraces $u \approx v$ means $u = xAy$, $v = xBCy$, where $A = B \cup C$, $B \cap C = \emptyset$, $B \times C \subseteq \text{ser}$.

1. It suffices to show that $u \approx v \implies u \div_R a \approx v \div_R a$. There are four cases:
 - (a) $a \in \Sigma(y)$. Let $z = y \div_R a$. Then $u \div_R a = xAz \approx xBCz = v \div_R a$.
 - (b) $a \notin \Sigma(y)$, $a \in A \cap C$. Then $u \div_R a = x(A \setminus \{a\})y \approx xB(C \setminus \{a\})y = v \div_R a$.
 - (c) $a \notin \Sigma(y)$, $a \in A \cap B$. Then $u \div_R a = x(A \setminus \{a\})y \approx x(B \setminus \{a\})Cy = v \div_R a$.
 - (d) $a \notin \Sigma(Ay)$. Let $z = x \div_R a$. Then $u \div_R a = zAy \approx zBCy = v \div_R a$.
2. Dually to (1).
3. It suffices to show that $u \approx v \implies \pi_D(u) \approx \pi_D(v)$. Note that $\pi_D(A) = \pi_D(B) \cup \pi_D(C)$, $\pi_D(B) \cap \pi_D(C) = \emptyset$ and $\pi_D(B) \times \pi_D(C) \subseteq \text{ser}$, so $\pi_D(u) = \pi_D(x)\pi_D(A)\pi_D(y) \approx \pi_D(x)\pi_D(B)\pi_D(C)\pi_D(y) = \pi_D(v)$. $\square$

Proposition 3 does not hold for an arbitrary absorbing monoid. For the absorbing monoid from Example 2 we have $u = \{a, b, c\} \equiv v = \{a\}\{b, c\}$, $u \div_R b = u \div_L b = \pi_{\{a, c\}}(u) = \{a, c\} \not\equiv \{a\}\{c\} = v \div_R b = v \div_L b = \pi_{\{a, c\}}(v)$.

Note that $(w \div_R a) \div_R b = (w \div_R b) \div_R a$, so we can define

$$w \div_R \{a_1, \dots, a_k\} \stackrel{\text{df}}{=} (\dots((w \div_R a_1) \div_R a_2) \dots \div_R a_k, \text{ and} \\ w \div_R A_1 \dots A_k \stackrel{\text{df}}{=} (\dots((c \div_R A_k) \div_R A_{k-1}) \dots \div_R A_1.$$

We define dually for $\div_L$.

Corollary 2. *For all $u, v, x \in \mathcal{S}$, we have*

1. $u \equiv v \implies u \div_R x \equiv v \div_R x$.
2. $u \equiv v \implies u \div_L x \equiv v \div_L x$.

□

The uniqueness of canonical representation for comtraces follows directly from the following result.

Lemma 1. *For each canonical step sequence $u = A_1 \dots A_k$, we have*

$$A_1 = \{a \mid \exists w \in [u]. w = C_1 \dots C_m \wedge a \in C_1\}.$$

Proof. Let $A = \{a \mid \exists w \in [u]. w = C_1 \dots C_m \wedge a \in C_1\}$. Since $u \in [u]$, $A_1 \subseteq A$. We need to prove that $A \subseteq A_1$. Definitely $A = A_1$ if $k = 1$, so assume $k > 1$. Suppose that $a \in A \setminus A_1$, $a \in A_j$, $1 < j \leq k$ and $a \notin A_i$ for $i < j$. Since $a \in A$, there is $v = Bx \in [u]$ such that $a \in B$. Note that $A_{j-1}A_j$ is also *canonical* and $u' = A_{j-1}A_j = (u \div_R (A_{j+1} \dots A_k)) \div_L (A_1 \dots A_{j-2})$. Let $v' = (v \div_R (A_{j+1} \dots A_k)) \div_L (A_1 \dots A_{j-2})$. We have $v' = B'x'$ where $a \in B'$. By Corollary 2, $u' \equiv v'$. Since $u' = A_{j-1}A_j$ is canonical then $\exists c \in A_{j-1}$. $(c, a) \notin \text{ser}$ or $\exists b \in A_j$. $(a, b) \notin \text{ser}$. For the former case: $\pi_{\{a, c\}}(u') = \{c\}\{a\}$ (if $c \notin A_j$) or $\pi_{\{a, c\}}(u') = \{c\}\{a, c\}$ (if $c \in A_j$). If $\pi_{\{a, c\}}(u') = \{c\}\{a\}$ then $\pi_{\{a, c\}}(v')$ equals either $\{a, c\}$ (if $c \in B'$) or $\{a\}\{c\}$ (if $c \notin B'$), i.e. in both cases $\pi_{\{a, c\}}(u') \neq \pi_{\{a, c\}}(v')$, contradicting Proposition 3(3). If $\pi_{\{a, c\}}(u') = \{c\}\{a, c\}$ then $\pi_{\{a, c\}}(v')$ equals either $\{a, c\}\{c\}$ (if $c \in B'$) or $\{a\}\{c\}\{c\}$ (if $c \notin B'$). However in both cases $\pi_{\{a, c\}}(u') \neq \pi_{\{a, c\}}(v')$, contradicting Proposition 3(3). For the latter case, let $d \in A_{j-1}$. Then $\pi_{\{a, b, d\}}(u') = \{d\}\{a, b\}$ (if $d \notin A_j$), or $\pi_{\{a, b, d\}}(u') = \{d\}\{a, b, d\}$ (if $d \in A_j$). If $\pi_{\{a, b, d\}}(u') = \{d\}\{a, b\}$ then $\pi_{\{a, b, d\}}(v')$ is one of the following $\{a, b, d\}$, $\{a, b\}\{d\}$, $\{a, d\}\{b\}$, $\{a\}\{b\}\{d\}$ or $\{a\}\{d\}\{b\}$, and in either case $\pi_{\{a, b, d\}}(u') \neq \pi_{\{a, b, d\}}(v')$, again contradicting Proposition 3(3). If $\pi_{\{a, b, d\}}(u') = \{d\}\{a, b, d\}$ then $\pi_{\{a, b, d\}}(v')$ is one of the following $\{a, b, d\}\{d\}$, $\{a, b\}\{d\}\{d\}$, $\{a, d\}\{b, d\}$, $\{a, d\}\{b\}\{d\}$, $\{a, d\}\{d\}\{b\}$, $\{a\}\{b\}\{d\}\{d\}$, $\{a\}\{d\}\{b\}\{d\}$, or $\{a\}\{d\}\{d\}\{b\}$. However in either case we have $\pi_{\{a, b, d\}}(u') \neq \pi_{\{a, b, d\}}(v')$, contradicting Proposition 3(3) as well. □

The above lemma does not hold for an arbitrary absorbing monoid. For both canonical representations of $\{[a, b]\{c\}\}$ from Example 9, namely $\{a, b\}\{c\}$ and $\{a, c\}\{b\}$, we have $A = \{a \mid \exists w \in [u]. w = C_1 \dots C_m \wedge a \in C_1\} = \{a, b, c\} \notin \mathcal{S}$. Adding A to $\mathcal{S}$ does not help as we still have $A \neq \{a, b\}$ and $A \neq \{a, c\}$.

Theorem 4. *For every comtrace $t \in \mathcal{S}^* / \equiv$ there exists exactly one canonical step sequence u such that $t = [u]$.*

Proof. The existence follows from Theorem 3. Suppose that $u = A_1 \dots A_k$ and $v = B_1 \dots B_m$ are both canonical step sequences and $u \equiv v$. By Lemma 1, we have $B_1 = A_1$. If $k = 1$, this ends the proof. Otherwise, let $u' = A_2 \dots A_k$ and $v' = B_2 \dots B_m$. By Corollary 2(2) we have $u' \equiv v'$. Since u' and v' are also canonical, by Lemma 1, we have $A_2 = B_2$, etc. Hence $u = v$. □

6 Relational Representation of Traces, Comtraces and Generalised Comtraces

It is widely known that Mazurkiewicz traces can represent partial orders. We show the similar relational equivalence for both comtraces and generalised comtraces.

6.1 Partial Orders and Mazurkiewicz Traces

Each trace can be interpreted as a partial order and each finite partial order can be represented by a trace. Let $t = \{x_1, \dots, x_k\}$ be a trace, and let $\prec_{x_i}$ be a total order defined by a sequence x_i , $i = 1, \dots, k$. The partial order generated by the trace t can then be defined as: $\prec_t = \bigcap_{i=1}^k \prec_{x_i}$. Moreover, the set $\{\prec_{x_1}, \dots, \prec_{x_n}\}$ is the set of all total extensions of $\prec_t$. Let X be a finite set, $\prec \subset X \times X$ be a partial order, $\{\prec_1, \dots, \prec_k\}$ be the set of all total extensions of $\prec$, and let $x_{\prec_i} \in X^*$ be a sequence that represents $\prec_i$, for $i = 1, \dots, k$. The set $\{x_{\prec_1}, \dots, x_{\prec_k}\}$ is a trace over the concurrent alphabet $(X, \curvearrowright)$.

6.2 Stratified Order Structures and Comtraces

Mazurkiewicz traces can be interpreted as a formal language representation of finite partial orders. In the same sense comtraces can be interpreted as a formal language representation of finite *stratified order structures*. Partial orders can adequately model “earlier than” relationship but cannot model “not later than” relationship [9]. Stratified order structures are pairs of relations and can model “earlier than” and “not later than” relationships.

A *stratified order structure* is a triple $Sos = (X, \prec, \sqsubset)$, where X is a set, and $\prec, \sqsubset$ are binary relations on X that satisfy the following conditions:

$$\begin{array}{ll} \text{C1:} & a \not\sqsubset a \\ \text{C2:} & a \prec b \implies a \sqsubset b \\ \text{C3:} & a \sqsubset b \sqsubset c \wedge a \neq c \implies a \prec c \\ \text{C4:} & a \sqsubset b \prec c \vee a \prec b \sqsubset c \implies a \prec c \end{array}$$

C1–C4 imply that $\prec$ is a partial order and $a \prec b \implies b \not\sqsubset a$. The relation $\prec$ is called “causality” and represents the “earlier than” relationship while $\sqsubset$ is called “weak causality” and represents the “not later than” relationship. The axioms C1–C4 model the mutual relationship between “earlier than” and “not later than” provided the system runs are defined as stratified orders.

Stratified order structures were independently introduced in [6] and [8] (the defining axioms are slightly different from C1–C4, although equivalent). Their comprehensive theory has been presented in [11]. It was shown in [10] that each comtrace defines a finite stratified order structure. The construction from [10] did not use the results of [11]. In this paper we present a construction based on the results of [11], which will be intuitively closer to the one used to show the relationship between traces and partial orders in Section 6.1.

Let $Sos = (X, \prec, \sqsubset)$ be a stratified order structure. A *stratified order* $\triangleleft$ on X is an *extension* of Sos if for all $a, b \in X$, $a \prec b \implies a \triangleleft b$ and $a \sqsubset b \implies a \triangleleft b$. Let $ext(Sos)$ denote the set of all extensions of Sos .

Let $u = A_1 \dots A_k$ be a step sequence. By $\bar{u} = \bar{A}_1 \dots \bar{A}_k$ be the *event enumerated* representation of t . We will skip a lengthy but intuitively obvious formal definition (see for example [10]), but for instance if $u = \{a, b\}\{b, c\}\{c, a\}\{a\}$, then $\bar{u} = \{a^{(1)}, b^{(1)}\}\{b^{(2)}, c^{(1)}\}\{a^{(2)}, c^{(2)}\}\{a^{(3)}\}$. Let $\Sigma_u = \bigcup_{i=1}^k \bar{A}_i$ denote the set of all enumerated events occurring in u , for $u = \{a, b\}\{b, c\}\{c, a\}\{a\}$, $\Sigma_u = \{a^{(1)}, a^{(2)}, a^{(3)}, b^{(1)}, b^{(2)}, c^{(1)}, c^{(2)}\}$. For each $\alpha \in \Sigma_u$, let $pos_u(\alpha)$ denote the consecutive number of a step where α belongs, i.e. if $\alpha \in \bar{A}_j$ then $pos_u(\alpha) = j$. For our example $pos_u(a^{(2)}) = 3$, $pos_u(b^{(2)}) = 2$, etc. For each enumerated even $\alpha = e^{(i)}$, let $l(\alpha)$ denote the *label* of α , i.e. $l(\alpha) = l(e^{(i)}) = e$. One can easily show ([10]) that $u \equiv v \implies \Sigma_u = \Sigma_v$, so we can define $\Sigma_{[u]} = \Sigma_u$.

Given a step sequence u , we define a stratified order $\triangleleft_u$ on Σ_u by: $\alpha \triangleleft_u \beta \iff pos_u(\alpha) < pos_u(\beta)$. Conversely, let $\triangleleft$ be a stratified order on a set X . The set X can be partitioned onto a unique sequence of non-empty sets $\Omega_{\triangleleft} = B_1 \dots B_k$ ($k \geq 0$) such that

$$\triangleleft = \bigcup_{i < j} (B_i \times B_j) \quad \text{and} \quad \simeq_{\triangleleft} = \bigcup_i (B_i \times B_i).$$

Unfortunately the proofs of two theorems below require introducing additional concepts and results, so we only provide sketches.

Theorem 5. *Let t be a comtrace over $(\mathcal{S}, sim, ser)$ and let $\prec_t, \sqsubset_t$ be two binary relations on Σ_t defined as:*

$$\begin{aligned} \alpha \prec_t \beta &\iff \forall u \in t. \alpha \triangleleft_u \beta, \\ \alpha \sqsubset_t \beta &\iff \forall u \in t. \alpha \triangleleft_u \beta. \end{aligned}$$

We have:

1. $Sos_t = (\Sigma_t, \prec_t, \sqsubset_t)$ is a stratified order structure,
2. $ext(Sos_t) = \{\triangleleft_u \mid u \in t\}$.

Proof. (sketch) The main part is to show that each stratified order $\triangleleft$ on Σ_t that satisfies: $\alpha \prec_t \beta \implies \alpha \triangleleft \beta$ and $\alpha \sqsubset_t \beta \implies \alpha \triangleleft \beta$ belongs to $\{\triangleleft_u \mid u \in t\}$. This can be done by induction on the number of steps of w , where w is the canonical step sequence such that $[w] = t$. The rest is a consequence of the results of [9,11]. $\square$

Theorem 6. *Let $Sos = (X, \prec, \sqsubset)$ be a stratified order structure, $\Delta = \{\Omega_{\triangleleft} \mid \triangleleft \in ext(Sos)\}$, $\mathcal{S}$ be the set of all steps of the step sequences in Δ , and let relations $sim, ser \subseteq l(X) \times l(X)$ be defined as follows:*

- $(l(\alpha), l(\beta)) \in sim \iff l(\alpha) \neq l(\beta) \wedge \exists A \in \mathcal{S}. \{l(\alpha), l(\beta)\} \subseteq A$,
- $(l(\alpha), l(\beta)) \in ser \iff (l(\alpha), l(\beta)) \in sim \wedge ((\alpha \sqsubset \beta \wedge \beta \not\sqsubset \alpha) \vee (\alpha \prec \beta))$.

Then we have:

1. $(\mathcal{S}, sim, ser)$ is a comtrace concurrent alphabet,
2. for each $u, v \in \Delta$ we have $u \equiv v$, i.e. $\Delta \in \mathcal{S}^*/\equiv$ is a comtrace.

Proof. (sketch) (1) is straightforward. To prove (2) we first take the canonical stratified extension of $\prec$ (see [10]), show that it belongs to $ext(Sos)$, and then show that it represents a canonical step sequence. Next we prove (2) by induction on the number of steps of this canonical step sequence. $\square$

6.3 Generalised Stratified Order Structures and Generalised Comtraces

A *generalised stratified order structure* is a triple $GSos = (X, \diamond, \sqsubset)$, where X is a non-empty set, and $\diamond, \sqsubset$ are two irreflexive relations on X , $\diamond$ is symmetric, and the triple $(X, \prec_G, \sqsubset)$, where $\prec_G = \diamond \cap \sqsubset$, is a stratified order structure (i.e. it satisfies C1–C4 from the previous subsection).

The relation $\diamond$ is called “commutativity” and represents the “earlier than or later than” relationship, while $\sqsubset$, called “weak causality” represents “not later than” relationship.

Generalised stratified order structures were introduced and their comprehensive theory has been presented in [7]. They can model any concurrent history when runs or observations are modelled by stratified orders (see [7]). We will show that each generalised comtrace defines a finite generalised stratified order structure and that each finite generalised stratified order structure can be represented by a generalised comtrace.

Let $GSos = (X, \diamond, \sqsubset)$ be a generalised stratified order structure. A *stratified order* $\triangleleft$ on X is an *extension* of $GSos$ if for all $a, b \in X$, $a \diamond b \implies a \triangleleft b$ or $b \triangleleft a$, and $a \sqsubset b \implies a \triangleleft b$. Let $ext(GSos)$ denote the set of all extensions of Sos .

Again the proofs of two theorems below require introducing additional concepts and results, so we only provide sketches.

Theorem 7. *Let t be a generalised comtrace over $(\mathcal{S}, sim, ser, inl)$ and let $\diamond_t, \sqsubset_t$ be two binary relations on Σ_t defined as:*

$$\begin{aligned} \alpha \diamond_t \beta &\iff \forall u \in t. (\alpha \triangleleft_u \beta \vee \beta \triangleleft_u \alpha), \\ \alpha \sqsubset_t \beta &\iff \forall u \in t. \alpha \triangleleft_u \beta. \end{aligned}$$

We have:

1. $GSos_t = (\Sigma_t, \diamond_t, \sqsubset_t)$ is a generalised stratified order structure,
2. $ext(GSos_t) = \{\triangleleft_u \mid u \in t\}$.

Proof. (sketch) The main part is to show that each stratified order $\triangleleft$ on Σ_t that satisfies: $\alpha \diamond_t \beta \implies \alpha \triangleleft \beta \vee \beta \triangleleft \alpha$ and $\alpha \sqsubset_t \beta \implies \alpha \triangleleft \beta$ belongs to $\{\triangleleft_u \mid u \in t\}$. This can be done by induction on the number of steps of w , where w is the canonical step sequence such that $[w] = t$ (we do not need a canonical representation to be unique here). The rest follows from the results of [7,11]. $\square$

Theorem 8. *Let $Sos = (X, \diamond, \sqsubset)$ be a generalised stratified order structure, $\Delta = \{\Omega_\triangleleft \mid \triangleleft \in ext(GSos)\}$, $\mathcal{S}$ be the set of all steps of the step sequences in Δ , and let relations $sim, ser, inl \subseteq l(X) \times l(X)$ be defined as follows:*

- $(l(\alpha), l(\beta)) \in sim \iff l(\alpha) \neq l(\beta) \wedge \exists A \in \mathcal{S}. \{l(\alpha), l(\beta)\} \subseteq A,$
- $(l(\alpha), l(\beta)) \in ser \iff (l(\alpha), l(\beta)) \in sim \wedge ((\alpha \sqsubset \beta \wedge \beta \not\sqsubset \alpha) \vee (\alpha \prec \beta)),$
- $(l(\alpha), l(\beta)) \in inl \iff (\forall A \in \mathcal{S}. l(\alpha) \notin A \vee l(\beta) \notin A) \wedge (\alpha \diamond \beta).$

Then we have:

1. $(\mathcal{S}, sim, ser, inl)$ is a generalised comtrace concurrent alphabet,
2. for each $u, v \in \Delta$ we have $u \equiv v$, i.e. $\Delta \in \mathcal{S}^*/\equiv$ is a generalised comtrace.

Proof. (sketch) (1) is straightforward. The proof of (2) is more complex than the proof of (2) of Theorem 6, as we need to show that there is a stratified order in $ext(GSos)$ which can be represented as an appropriate canonical step sequence (no uniqueness needed). Next we prove (2) by induction on the number of steps of this canonical step sequence. $\square$

7 Paradigms of Concurrency

The general theory of concurrency developed in [9] provides a hierarchy of models of concurrency, where each model corresponds to a so called “paradigm”, or a general rule about the structure of concurrent histories, where concurrent histories are defined as sets of equivalent partial orders representing particular system runs. In principle, a paradigm describes how simultaneity is handled in concurrent histories. The paradigms are denoted by π_1 through π_8 . It appears that only paradigms π_1 , π_3 , π_6 and π_8 are interesting from the point of view of concurrency theory. The paradigms were formulated in terms of partial orders. Comtraces are sets of step sequences, each step sequence uniquely defines a stratified order, so the comtraces can be interpreted as sets of equivalent partial orders, i.e. concurrent histories (see [10] for details). The most general paradigm, π_1 , assumes no additional restrictions for concurrent histories, so each comtrace conforms trivially to π_1 . The paradigms π_3 , π_6 and π_8 , when translated into the comtrace formalism, impose the following restrictions.

Let (E, sim, ser, inl) be a generalised comtrace alphabet. The monoid of generalised comtraces (comtraces when $inl = \emptyset$) $(\mathcal{S}^*/\equiv, \hat{\circ}, [\lambda])$ conforms to:

$$\begin{aligned} \text{paradigm } \pi_3 &\iff \forall a, b \in E. (\{a\}\{b\} \equiv \{b\}\{a\} \Rightarrow \{a, b\} \in \mathcal{S}). \\ \text{paradigm } \pi_6 &\iff \forall a, b \in E. (\{a, b\} \in \mathcal{S} \Rightarrow \{a\}\{b\} \equiv \{b\}\{a\}). \\ \text{paradigm } \pi_8 &\iff \forall a, b \in E. (\{a\}\{b\} \equiv \{b\}\{a\} \Leftrightarrow \{a, b\} \in \mathcal{S}). \end{aligned}$$

Proposition 4. 1. Every monoid of comtraces conforms to π_3 .

2. If π_8 is satisfied then $ind = ser = sim$.

Proof. 1. Let $\{a\}\{b\} \equiv \{b\}\{a\}$ for some $a, b \in E$. This means $\{a\}\{b\} \approx^{-1} \{a, b\} \approx \{b\}\{a\}$, i.e. $\{a, b\} \in \mathcal{S}$.

2. Clearly $ind \subseteq ser \subseteq sim$. Let $(a, b) \in sim$. This means $\{a, b\} \in \mathcal{S}$, which, by π_8 , implies $\{a\}\{b\} \equiv \{b\}\{a\}$, i.e. $(a, b) \in ind$. $\square$

From Proposition 4 it follows that comtraces cannot model any concurrent behaviour (history) that does not conform to the paradigm π_3 . Generalised comtraces conform only to π_1 , so they can model any concurrent history that is represented by a set of equivalent step sequences.

If a monoid of comtraces conforms to π_6 it also conforms to π_8 . Proposition 4 says all comtraces conforming to π_8 can be reduced to equivalent Mazurkiewicz traces.

8 Conclusion

The concepts of absorbing monoids over step sequences, partially commutative absorbing monoids over step sequences, absorbing monoids with compound generators, and monoids of generalised comtraces have been introduced and

analysed. They all are generalisations of Mazurkiewicz traces [5] and comtraces [10]. Some new properties of comtraces and their relationship to stratified order structures [11] have been discussed. The relationship between generalised comtraces and generalised stratified order structures [7] was also analysed.

Despite some obvious advantages, for instance, very handy composition and no need to use labels, quotient monoids (perhaps with the exception of Mazurkiewicz traces) are much less popular in dealing with issues of concurrency than their relational counterparts partial orders, stratified order structures, occurrence graphs, etc. We believe that in many cases, quotient monoids could provide simpler and more adequate models of concurrent histories than their relational equivalences.

Acknowledgement. The authors thanks all four referees for a number of very detailed and helpful comments.

References

1. Cartier, P., Foata, D.: Problèmes combinatoires de commutation et réarrangements. Lecture Notes in Mathematics, vol. 85. Springer, Heidelberg (1969)
2. Cohn, P.M., Reidel, D.: Universal Algebra (1981)
3. Davillers, R., Janicki, R., Koutny, M., Lauer, P.E.: Concurrent and Maximally Concurrent Evolution of Non-sequential Systems. Theoretical Computer Science 43, 213–238 (1986)
4. Desel, J.: Private Information, Communicated to the authors by G. Juhás (2007)
5. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
6. Gaifman, H., Pratt, V.: Partial Order Models of Concurrency and the Computation of Function. In: Proc. of LICS 1987, pp. 72–85. IEEE, Los Alamitos (1987)
7. Guo, G., Janicki, R.: Modelling Concurrent Behaviours by Commutativity and Weak Causality Relations. In: Kirchner, H., Ringissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 178–191. Springer, Heidelberg (2002)
8. Janicki, R., Koutny, M.: Invariants and Paradigms of Concurrency Theory. In: Aarts, E.H.L., van Leeuwen, J., Rem, M. (eds.) PARLE 1991. LNCS, vol. 506, pp. 59–74. Springer, Heidelberg (1991)
9. Janicki, R., Koutny, M.: Structure of Concurrency. Theoretical Computer Science 112(1), 5–52 (1993)
10. Janicki, R., Koutny, M.: Semantics of Inhibitor Nets. Information and Computation 123(1), 1–16 (1995)
11. Janicki, R., Koutny, M.: Fundamentals of Modelling Concurrency Using Discrete Relational Structures. Acta Informatica 34, 367–388 (1997)
12. Juhás, G., Lorenz, R., Mauser, S.: Synchronous + Concurrent + Sequential = Earlier Than + Not Later Than. In: Proc. of ACSD 2006 (Application of Concurrency to System Design), Turku, Finland, pp. 261–272. IEEE Press, Los Alamitos (2006)
13. Kleijn, H.C.M., Koutny, M.: Process Semantics of General Inhibitor Nets. Information and Computation 190, 18–69 (2004)
14. Mazurkiewicz, A.: Introduction to Trace Theory, in [5], pp. 3–42
15. Ochmański, E.: Recognizable Trace Languages, in [5], pp. 167–204
16. Shields, M.W.: Adequate Path Expressions. In: Kahn, G. (ed.) Semantics of Concurrent Computation. LNCS, vol. 70, pp. 249–265. Springer, Heidelberg (1979)

Labeled Step Sequences in Petri Nets

Matthias Jantzen and Georg Zetsche

University of Hamburg, Faculty of Mathematics, Informatics, and Natural Sciences,
Department of Informatics
Vogt-Kölln-Straße 30, 22527 Hamburg
`jantzen@informatik.uni-hamburg.de`, `3zetzsch@informatik.uni-hamburg.de`

Abstract. We compare various modes of firing transitions in Petri nets and investigate classes of languages specified by them. We define languages through steps, (i.e., sets of transitions), maximal steps, multi-steps, (i.e., multisets of transitions), and maximal multi-steps of transitions in Petri nets. However, by considering labeled transitions, we do this in a different manner than in [Burk 81a, Burk 83]. Namely, we allow only sets and multisets of transitions to form a (multi-)step, if they all share the same label. In a sequence of (multi-)steps, each of them contributes its label once to the generated word. Through different firing modes that allow multiple use of transitions in a single multi-step, we obtain a hierarchy of families of languages. Except for the maximal multi-steps all classes can be simulated by sequential firing of transitions.

1 Introduction

Classes of languages that reflect an interleaving semantics are usually defined by Petri nets as sets of sequences of labeled or unlabeled transitions. These classes, which have been studied since 1972 by Baker, [Bak 72], have also been extensively investigated by [Hack 76], [Grei 78], [Jant 79a], [Pete 81] and others. Steps as a description of concurrent application of transitions have been studied already in the beginning of the development of Petri nets by H. Genrich, K. Lautenbach and C.A. Petri. In connection with computation the concept of maximal parallelity was suggested by Salwicki and Müldner [SaMu 81] and apparently was first studied by Burkhard in [Burk 80, Burk 81a, Burk 83].

A completely different viewpoint on the parallel firing of transitions in Petri nets was taken by Farwer, Kudlek and Rölke in [FaKR 06], where each token within a marking is associated with an individual read/write head on a tape of a so-called Concurrent Turing Machine (CTM). The variant of a Concurrent Finite Automaton (CFA) has been defined and studied in [JaKZ 07a], [JaKZ 07b, JaKZ 08] and [FJKRZ 07]. Step transition systems in connection with Petri nets have also been studied under a categorical view in [Muku 92]. Steps are studied in connection with bisimulation in [NiPS 95].

We distinguish two kinds of multi-steps: steps, which contain every transition at most once, and general multi-steps, which are arbitrary multisets of transitions. Completely new here, is the way we denote labeled steps and sequences

of steps: We allow only sets and multisets of transitions to form a (multi-)step, if all transition share the same label and then use this label just once for each (multi-)step, when creating a sequence of them.

Another structure that describes the behaviour of Petri nets is given by Mazurkiewicz traces, [DiRo 95]. The concurrent application of transitions is thereby represented by commuting symbols in the generated language. We, however, do not propagate the number of concurrently applied transitions. Instead, any (multi-)set of concurrently firing transitions, all of which carry the same label, results in one symbol in the generated word.

In [Kudl 05], Kudlek has used several different strategies of sequential, concurrent or parallel usage of transition firings to define classes of reachability sets, including those obtainable by the maximal firing discipline from Burkhard, [Burk 80]. The maximum strategy for multi-steps, which appear as the multiset sum of several steps, is related to maximal steps in condition event systems and was suggested by the MAX-semantics for concurrent computations by Salwicki and Müldner, [SaMu 81]. These considerations lead to an extended computational power and showed the existence of parallel programs, which cannot be faithfully represented by single processor computations. Burkhard has also defined languages using Petri net steps by writing down all permutations of the transitions that form such a step. He thereby defines the Petri net language in an interleaving semantics, which does not directly reflect the parallel use of transition firings. In [Kudl 05], the focus was put on reachability sets and languages were not considered at all.

In this paper, we will show that parallel firings of equally labeled transition add more power to Petri nets while still keeping decidability for many, though not all, questions. Also, it was now possible to compare these (multi-)step sequences directly with the well known families in the Chomsky hierarchy. In short, we add a number of missing aspects of languages defined by labeled Petri net steps.

2 Preliminaries

In describing and using Petri nets, there are two common variants to denote markings in Petri nets: In Place/Transition nets (P/T-nets), having black tokens only, a marking m is usually an element of $\mathbb{N}^k$, where k is the number of places in the P/T-net under consideration. In colored Petri nets, tokens can have different colors and a marking of a single place is denoted by a multiset over the color domain. We use multisets of places to denote a marking of a P/T-net.

Definition 1. *A set M with an operation $+$: $M \times M \rightarrow M$ is a monoid, if the operation is associative and there is a neutral element $0 \in M$ for which $0 + x = x + 0 = x$ for every $x \in M$. M is called commutative, if $x + y = y + x$ for all $x, y \in M$. For $x, y \in M$, let $x \sqsubseteq y$ iff there is a $z \in M$ with $y = x + z$.*

For every set A , we have the set $A^\oplus$ of mappings $\mu : A \rightarrow \mathbb{N}$. The elements of $A^\oplus$ are called multisets over A . The multiset μ is a set, iff $\forall a \in A : \mu(a) \leq 1$.

With the operation $\oplus$, defined by $(\mu \oplus \nu)(a) = \mu(a) + \nu(a)$, $A^\oplus$ becomes a commutative monoid with elements written in additive notation. The neutral element is the empty multiset and will be denoted by $\emptyset$, with $\emptyset(a) := 0$ for every $a \in A$.

The concrete multiset $\nu \in \{a, b, c\}^\oplus$ with $\nu(a) = 3, \nu(b) = 0$, and $\nu(c) = 1$ will be written as $3a \oplus c$, so that singletons are denoted by the elements alone, omitting the factor 1.

Whether an element $a \in A$ has to be considered as a singleton multiset $a \in A^\oplus$ or as the mere element of the set A , will become clear from the context. Other known notions for the multiset $3a \oplus c$, as for instance $\{a, a, a, c\}$ or a^3c will not be used in this work.

We also use the nonnegative integers $\mathbb{N}$ as factors of multisets $\nu \in A^\oplus$ (usually from the left) and define $\forall x \in A : \forall i \in \mathbb{N} : (i \cdot \nu)(x) := i \cdot (\nu(x))$. Together with the factors from the semiring $[\mathbb{N}, +, \cdot, 0, 1]$ the structure of multisets $A^\oplus$ is a $\mathbb{N}$ -semimodule, see [KuSa 86].

For finite sets A , we observe an important property of monoids of the form $A^\oplus$: For every mapping $f : A \rightarrow B$, there is an induced mapping $f^* : A^\oplus \rightarrow B^\oplus$, defined by

$$f^*(\mu)(b) := \bigoplus_{a \in A, f(a)=b} \mu(a),$$

for every $\mu \in A^\oplus$ and $b \in B$.

We define $(\nu \ominus \mu)(a) := \max(0, \nu(a) - \mu(a))$ for all $a \in A$ which is equivalent to set difference, in case the multisets μ, ν describe sets.

A multiset ν is called finite, iff $\{a \in A \mid \nu(a) \neq 0\}$ is finite, and then let $|\nu| := \sum_{a \in A} \nu(a)$. These definitions are carried over to $\mathbb{N}^k$ by noting that $\mathbb{N}^k \cong \{a_1, \dots, a_k\}^\oplus$. The summation $\sum_{i \in I} \nu_i$ is interpreted component-wise, in case the elements ν_i are vectors from $\mathbb{N}^k$, and uses the multiset addition $\oplus$, if the elements ν_i are arbitrary multisets.

If Σ is an alphabet, then Σ^+ denotes the set of all non-empty strings over Σ and $\Sigma^* := \Sigma^+ \cup \{\lambda\}$ is the free monoid generated by Σ with the empty string λ as neutral element. Since we will often use it, we define $\Sigma^\lambda := \Sigma \cup \{\lambda\}$. For any $a \in \Sigma$ and each $w \in \Sigma^*$, let $|w|_a$ denote the number of occurrences of the symbol a within w so that $|w| := \sum_{a \in \Sigma} |w|_a$ denotes the length of w .

Definition 2. A labeled Petri net is a 5-tuple $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, where

- Σ is an alphabet.
- $N = (P, T, \partial_0, \partial_1)$ is a Petri net, where P and T are the finite, non-empty sets of places and transitions, respectively. $\partial_0, \partial_1 : T^\oplus \rightarrow P^\oplus$ are homomorphisms that specify the pre- and post-multisets. We require $\forall t \in T : \partial_0(t) \neq \emptyset$, since otherwise maximal multi-steps (see Def. 4) cannot be defined. The basic elements $\partial_0(t)$ and $\partial_1(t)$ are specified by the graphical representation of the Petri net: $\forall p \in P : \partial_0(t)(p)$ gives the number of directed arcs that exist from p to t , while $\forall p \in P : \partial_1(t)(p)$ is the number of directed arcs that point from t to p .

- $\sigma : T \rightarrow \Sigma^\lambda$ defines the symbol-wise labeling for every transition. A transition $t \in T$ is called λ -transition, if $\sigma(t) = \lambda$.
- $m_0 \in P^\oplus$ is the initial marking and $\mathcal{F} \subseteq P^\oplus$ is a finite set of final markings.

In order to use Petri nets as specification of formal languages, the first definitions were given by [Bake 72], [Hack 76], [Grei 78], [Jant 79a], [Pete 81] who considered the labeled firing sequences. In the following, we adapt these definitions to various types of step sequences.

Definition 3. Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, $N = (P, T, \partial_0, \partial_1)$, be a labeled Petri net.

- For any two markings $m_1, m_2 \in P^\oplus$ and transition $t \in T$, write $m_1 \xRightarrow[\sigma(t)]{} m_2$ if $\partial_0(t) \sqsubseteq m_1$ and $m_2 = m_1 \ominus \partial_0(t) \oplus \partial_1(t)$. This is recursively generalized to (labeled) firing sequences: Let $w \in \Sigma^*$, $t \in T$ and $m_1, m_3 \in P^\oplus$ then $m_1 \xRightarrow[w\sigma(t)]{} m_3$ iff $\exists m_2 \in P^\oplus : m_1 \xRightarrow[w]{} m_2 \xRightarrow[\sigma(t)]{} m_3$. In addition, let $m \xRightarrow[\lambda]{} m$ for any marking $m \in P^\oplus$.
- According to Hack, the (terminal) Petri net language is $L(K) := \{w \in \Sigma^* \mid \exists m_f \in \mathcal{F} : m_0 \xRightarrow[w]{} m_f\}$.
 $L(K)$ is called λ -free labeled, iff $\sigma(t) \neq \lambda$ for each $t \in T$.
- The family of arbitrarily labeled (resp. λ -free labeled) Petri net languages is denoted by $\mathcal{L}_0^\lambda$ ($\mathcal{L}_0$, resp.).

An important Petri net language is the semi-Dyck language over one pair of brackets $D_1 := \{w \in \{x, \bar{x}\}^* \mid |w|_x = |w|_{\bar{x}} \wedge w = uv \implies |u|_x \geq |u|_{\bar{x}}\}$, which generates the families $\mathcal{L}_0^\lambda$ and $\mathcal{L}_0$ as least intersection-closed (full) trio: $\mathcal{L}_0 = \mathcal{M}_\cap(D_1)$ and $\mathcal{L}_0^\lambda = \hat{\mathcal{M}}_\cap(D_1)$, see [Jant 79a, Grei 78].

3 Definitions

Steps in Petri nets are sets of transitions that fire independently and parallel at the same time, even parallel to themselves.

We, however, distinguish the class of classical steps into those which are sets of transitions and those which are multisets of transitions (that are not necessarily sets).

If every transition occurs only once, like in elementary net systems, thus having single usage within the step, then we use the naming step. A multiset of transition, on the other hand, may contain more than one occurrence of a transition. Since such multisets of transitions can be seen as the sum of several single steps, they will shortly be called multi-steps.

In both cases, we only consider (multi-)steps of transitions having the same label. The labeled step sequences, multi-step sequences and maximal multi-step sequences are then defined accordingly:

Definition 4. Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, $N = (P, T, \partial_0, \partial_1)$, be a labeled Petri net.

- For $x \in \Sigma^\lambda$ let $T_x := \{t \in T \mid \sigma(t) = x\}$.
- For any two markings $m_1, m_2 \in P^\oplus$, $x \in \Sigma^\lambda$, we write $m_1 \xRightarrow{x}_m m_2$ iff $\exists \nu \in (T_x)^\oplus, \nu \neq \emptyset$, such that $\partial_0(\nu) \sqsubseteq m_1$ and $m_2 = m_1 \ominus \partial_0(\nu) \oplus \partial_1(\nu)$. Such a multi-step is called maximal, iff $\forall \mu \in (T_x)^\oplus : \nu \sqsubseteq \mu$ and $\partial_0(\mu) \sqsubseteq m_1$ imply $\nu = \mu$.

If $\nu \in (T_x)^\oplus$ is actually a set, that is $\nu \subseteq T_x$, we will only name it a ‘step’ and signalize this by writing $m_1 \xRightarrow{x}_s m_2$. A step is called maximal, iff $\forall \mu \subseteq T_x : \nu \sqsubseteq \mu$ and $\partial_0(\mu) \sqsubseteq m_1$ imply $\nu = \mu$.

In case $\nu \in (T_x)^\oplus$ is a maximal multi-step, we write $m_1 \xRightarrow{x}_{\hat{m}} m_2$, and $m_1 \xRightarrow{x}_s m_2$, in case we consider maximal steps. Note, maximality is defined for steps wrt. steps and for multi-steps wrt. multi-steps, so that a step may be maximal with respect to all steps but not maximal with respect to all multi-steps! All four notions are recursively generalized to (labeled) step sequences, multi-step sequences, maximal step sequences, and maximal multi-step sequences:

- Let $w \in \Sigma^*$, $x \in \Sigma^\lambda$, $\nu \in (T_x)^\oplus$, $\alpha \in \{s, m, \hat{s}, \hat{m}\}$, and $m_1, m_3 \in P^\oplus$. Then $m_1 \xRightarrow{wx}_\alpha m_3$ iff $\exists m_2 \in P^\oplus : m_1 \xRightarrow{w}_\alpha m_2 \xRightarrow{x}_\alpha m_3$. In addition, let $m \xRightarrow{\lambda}_\alpha m$ for any marking $m \in P^\oplus$ and $\alpha \in \{s, m, \hat{s}, \hat{m}\}$.

If a multi-step μ is maximal, then it means, that no strictly larger multi-step is enabled in that marking. A maximal multi-step need not be unique: there may well be other maximal multi-steps enabled. However, there are only finitely many of them and they are all incomparable to μ with respect to multiset inclusion $\sqsubseteq$.

Definition 5. Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, $N = (P, T, \partial_0, \partial_1)$, be a labeled Petri net.

- The terminal languages of arbitrarily labeled steps, of arbitrarily labeled multi-steps, of arbitrarily labeled maximal steps, and of arbitrarily labeled maximal multi-steps are:

$$L_\alpha(K) := \{w \in \Sigma^* \mid \exists m_f \in \mathcal{F} : m_0 \xRightarrow{w}_\alpha m_f\}, \text{ for } \alpha \in \{s, m, \hat{s}, \hat{m}\}.$$

A language $L_\alpha(K)$, $\alpha \in \{s, m, \hat{s}, \hat{m}\}$, is called λ -free labeled, iff $\sigma(t) \neq \lambda$ for each $t \in T$.

- For each $\alpha \in \{s, m, \hat{s}, \hat{m}\}$ we denote the family of arbitrarily labeled (resp. λ -free labeled) terminal multi-step sequence language of the corresponding type by $\mathcal{L}_\alpha^\lambda$ ($\mathcal{L}_\alpha$, resp.).

From the definitions and a simple construction, one easily gets the following inclusions:

Lemma 1. $\mathcal{L}_0 \subseteq \mathcal{L}_0^\lambda, \mathcal{L}_s \subseteq \mathcal{L}_s^\lambda, \mathcal{L}_m \subseteq \mathcal{L}_m^\lambda, \mathcal{L}_{\hat{s}} \subseteq \mathcal{L}_{\hat{s}}^\lambda, \mathcal{L}_{\hat{m}} \subseteq \mathcal{L}_{\hat{m}}^\lambda,$
 $\mathcal{L}_0 \subseteq \mathcal{L}_s, \mathcal{L}_0 \subseteq \mathcal{L}_m, \mathcal{L}_0^\lambda \subseteq \mathcal{L}_s^\lambda, \text{ and } \mathcal{L}_0^\lambda \subseteq \mathcal{L}_m^\lambda.$

Proof. The first five inclusions follow immediately from the definition. The remaining inclusions can be shown as follows:

Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$ be a labeled Petri net. In order to obtain $L(K)$ as a set $L_s(K')$ of labeled step (or multi-step) sequences for a Petri net K' we modify the Petri net $N = (P, T, \partial_0, \partial_1)$ by adding a so-called run-place $r \notin P$ to yield $N' := (P', T, \partial'_0, \partial'_1)$, where $P' := P \cup \{r\}$ and the pre- and post mappings ∂'_0 and ∂'_1 are modified as follows: $\forall t \in T : \partial'_0(t) := \partial_0(t) \oplus \{r\} \wedge \partial'_1(t) := \partial_1(t) \oplus \{r\}$. Finally, the initial and final markings have to be extended by one token on the place r . The labeling mapping σ remains unaltered. Since the run-place has exactly one token and self-loops with each transition, any possible step (and multi-step) occurring in N' consists of exactly one transition. Hence, the labeled step and multi-step sequences in K' are precisely the labeled transition sequences of K , i.e., $L_s(K') = L_m(K') = L(K)$. $\square$

The technique used by Kudlek, [Kudl 05], to show that the family of reachability sets of ordinary Petri nets with single usage of the transitions coincides with the family of reachability sets of Petri nets under the steps firing mode, can be used to show a similar result for the language classes. Labeled step sequence languages of Petri nets coincide with ordinary Petri net languages. We show this here only for languages defined by final markings to be reached. The classes of covering or deadlock languages defined by Peterson, [Pete 81], will not be considered here, but may be defined for step sequences, as well.

Lemma 2.

$$\mathcal{L}_0 = \mathcal{L}_s \subsetneq \mathcal{L}_0^\lambda = \mathcal{L}_s^\lambda$$

Proof. $\mathcal{L}_0 \subseteq \mathcal{L}_s$ and $\mathcal{L}_0^\lambda \subseteq \mathcal{L}_s^\lambda$ is known from Lemma 1. To show the converse, we begin with a labeled Petri net $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$ with its language of labeled step sequences $L_s(K)$ and construct a new labeled Petri net $K' = (\Sigma, N', \sigma', m_0, \mathcal{F})$ with $N' := (P, T', \partial'_0, \partial'_1)$ by adding new transitions t_ν for all possible steps $\nu \subseteq T_x$ getting the labeling $\sigma'(t_\nu) := x$. The pre- and post mappings of t_ν are defined by $\partial'_0(t_\nu) := \sum_{t \in \nu} \partial_0(t)$ and $\partial'_1(t_\nu) := \sum_{t \in \nu} \partial_1(t)$.

It is now obvious, that $L(K') = L_s(K)$ for any labeling mapping σ . Hence we obtain $\mathcal{L}_0 = \mathcal{L}_s$ and $\mathcal{L}_0^\lambda = \mathcal{L}_s^\lambda$.

The strict inclusion $\mathcal{L}_0 \subsetneq \mathcal{L}_0^\lambda$ was shown in [Grei 78] and independently in [Jant 79a]. $\square$

We can simulate a (maximal) step by a (maximal) multi-step, if we restrict each transition to a single firing. This can be guaranteed by adding a marked place r_t to each transition, which self-loops with that transition, i.e., p_t is added to $\partial_0(t)$,

as well as to $\partial_1(t)$. Also, the multiset sum $\sum_{t \in T} r_t$ has to be added to the initial and to the final marking of the labeled Petri net. Without giving a rigorous proof we see:

Lemma 3.

$$\mathcal{L}_s \subseteq \mathcal{L}_m \quad \text{and} \quad \mathcal{L}_{\hat{s}} \subseteq \mathcal{L}_{\hat{m}}$$

If we use exactly one and the same run-place for all transitions, i.e. add a place r which self-loops with each transition, then any maximal step consists of a single transition only, and we have:

Lemma 4.

$$\mathcal{L}_0 = \mathcal{L}_s \subseteq \mathcal{L}_{\hat{s}}$$

Lemma 5.

$$\mathcal{L}_m \subseteq \mathcal{L}_{\hat{m}},$$

Proof. Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$ with $N = (P, T, \partial_0, \partial_1)$ be a labeled Petri net. We construct a new labeled Petri net $K' = (\Sigma, N', \sigma', m'_0, \mathcal{F}')$ with $N' = (P', T', \partial'_0, \partial'_1)$ as follows: First we add two places r and q , where q is a failure place which will never be emptied. This place q is not contained in any other, especially final, marking. The place r is initially marked and will be added to the initial and final markings, hence $P' := P \cup \{r, q\}$.

The new transitions $t_x, x \in \Sigma^\lambda$, labeled $\sigma'(t_x) := x$ are added to T , and may transfer the token from r to q : $\forall x \in \Sigma^\lambda : \partial'_0(t_x) := r, \partial'_1(t_x) := q$. This will guarantee, that any maximal step must avoid these transitions and has to use exactly one of those in $\tilde{T}$, defined next:

$\tilde{T} := \{t' \mid t \in T\}$, with $\partial'_i(t') := r \oplus \partial_i(t)$, for $i \in \{0, 1\}$, and $\sigma'(t') := \sigma(t)$. In order to extend any multi-step to a maximal step we add for each place, and each $x \in \Sigma^\lambda$, simple transitions that do not change the markings and only loop with its corresponding places:

$\bar{T} := \{t_{x,p} \mid x \in \Sigma^\lambda \wedge p \in P\}$, with $\partial'_0(t_{x,p}) := \partial'_1(t_{x,p}) := p$, and $\sigma'(t_{x,p}) := x$. Finally: $T' := T \cup \{t_x \mid x \in \Sigma^\lambda\} \cup \tilde{T} \cup \bar{T}$. It can be seen, that any maximal step in K' that does not mark the failure place q , corresponds to some multi-step in K . Conversely, to any multi-step in K there is an equally labeled maximal multi-step in K' . $\square$

However, if we compare steps with multi-steps we recognize a big difference:

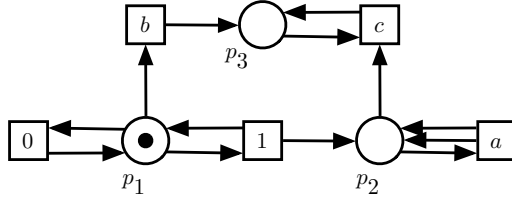
Theorem 1.

$$\mathcal{L}_s \subsetneq \mathcal{L}_m$$

Proof. The inclusion has been shown in Lemma 3, so only the inequality has to be proved:

Let N_{bin} be the following labeled Petri net (Fig.1) with initial marking p_1 , final marking p_3 and the identity as the labeling function. Consider only labeled multi-step sequences from the set

$$R := (\{0, 1\}\{a\})^*\{0, 1\}\{b\}\{c\}^*, \quad (1)$$

Fig. 1. N_{bin}

where each use of the transition labeled a is a single (up to the largest) possible multi-step in the marking reached. Among many other multi-step sequences outside the set R , the following are the only ones possible within R : $p_1 \xrightarrow[w]{*} p_3$, $w = i_n a i_{n-1} a \cdots a i_0 b c^k$, where the symbols $i_j \in \{0, 1\}$ are interpreted as binary digits, $[i_n \cdots i_0]_2$ denotes the integer represented by $i_n \cdots i_0$ as binary number, and $|w|_1 \leq k = |w|_c \leq [i_n \cdots i_0]_2$. If $L_m(N_{bin}) \in \mathcal{L}_s$, then $L_{bin} := R \cap L_m(N_{bin}) \in \mathcal{L}_0 = \mathcal{L}_s$ since $\mathcal{L}_0 = \mathcal{M}_\cap(D_1)$, [Jant 79a], is closed w.r.t. intersection by regular sets.

However, $L_{bin} \notin \mathcal{L}_0$ can be shown by almost the same technique that was used to show $\mathcal{L}_0 \subsetneq \mathcal{L}_0^\lambda$ in [Jant 79a], Theorem 3 or in [Grei 78], by observing, that there are only polynomially many different reachable markings when reading a string from L_{bin} up to the symbol b , while $O(2^n)$ are necessary. This technique is used and explained also in the proof of Theorem 2.

Hence, $L_{bin} = \{w \in R \mid w = i_n a i_{n-1} a \cdots a i_0 b c^k \wedge |w|_1 \leq k \leq [i_n \cdots i_0]_2\} \notin \mathcal{L}_s$ implies $L_m(N_{bin}) \notin \mathcal{L}_s = \mathcal{L}_0$, and $\mathcal{L}_s \subsetneq \mathcal{L}_m$ follows. $\square$

The polynomial bound on the number of different reachable markings in transition sequences of length n can also be observed if sequences of length n with maximal steps are used. This yields a similar result by a variation of the well known proof technique.

Theorem 2.

$$\mathcal{L}_{\hat{s}} \subsetneq \mathcal{L}_{\hat{m}}$$

Proof. We already know the inclusion $\mathcal{L}_{\hat{s}} \subseteq \mathcal{L}_{\hat{m}}$ from Lemma 3, so we have to prove its strictness. Let N_{bin} be the labeled Petri net used in the proof of Theorem 1 (see Fig. 1). Further, define R as in (1). It is easy to see, that the class $\mathcal{L}_{\hat{m}}$ is closed against regular intersection. Therefore, $\mathcal{L}_{\hat{s}} = \mathcal{L}_{\hat{m}}$ would imply $\mathcal{L}_{\hat{m}} \ni L_{\hat{m}}(N_{bin}) \cap R = L_{bin} \in \mathcal{L}_{\hat{s}}$.

Assume that $L_{bin} = L_{\hat{s}}(K)$ for some labeled Petri net $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, $N = (P, T, \partial_0, \partial_1)$. Then, in each single step of a step-sequence in N at most $|T|$ transitions may fire concurrently. Let $k_s := |T| \cdot \max \{|\partial_1(t)| \mid t \in T\}$. We see, that in a computation starting at m_0

$$m_0 \xrightarrow[i_1 a]{*}_s m_1 \xrightarrow[i_2 a]{*}_s m_2 \cdots \xrightarrow[i_{n-1} a]{*}_s m_{n-1} \xrightarrow[i_n b]{*}_s m_n,$$

with $i_j \in \{0, 1\}$, we have $\forall i \leq n : |m_i| \leq k_s^{2i} \cdot |m_0|$. The maximal number of different markings obtainable within a maximal step-sequence of length $2n$ that ends in firing the b -transition is

$$\binom{|m_0| + k_s \cdot 2n + |P|}{|P|} \leq (|m_0| + k_s \cdot 2n + 1)^{|P|},$$

a polynomial in n . But as in the proof of $\mathcal{L}_0 \neq \mathcal{L}_0^\lambda$ in [Grei 78, Jant 79a], see Theorem 1, there should be at least $O(2^n)$ markings, since there are that many different maximal step-sequences of length $2n$ that end in firing the b -transition. For large n this is impossible. □

This technique of proof can also be used to show that certain context-free languages cannot be elements of the class $\mathcal{L}_{\hat{s}}$:

Lemma 6.

$$\mathcal{L}_s \subseteq \mathcal{L}_{\hat{s}} \not\subseteq \mathcal{Cf} \text{ and } \mathcal{Cf} \not\subseteq \mathcal{L}_{\hat{s}}$$

Proof. Let $L_{\text{dcopy}} := \{wcu^{\text{rev}} \mid w \in \{a, b\}^*\} \in \mathcal{Cf}$. There are 2^n different prefixes wc for strings in L_{dcopy} of length $2n + 1$. Hence, if this language should be created by a labeled Petri net K as $L_s(K)$ or as $L_{\hat{s}}(K)$ then the number of possible different markings reached after reading words of length $n + 1$ in K is polynomial in n , thus $L_{\text{dcopy}} \notin \mathcal{L}_{\hat{s}}$.

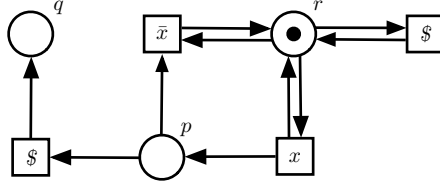
Conversely, it is easily observed, that there are Petri net languages in the class $\mathcal{L}_s = \mathcal{L}_0$ that are not context free. Hence the classes $\mathcal{L}_{\hat{s}}$ and $\mathcal{Cf}$ are incomparable. For the classes $\mathcal{Cf}$ and $\mathcal{L}_s = \mathcal{L}_0$ incomparability is already known from [Grei 78, Jant 79a]. □

Lemma 7.

$$\mathcal{L}_s \not\subseteq \mathcal{L}_{\hat{s}}, \quad \mathcal{L}_m \not\subseteq \mathcal{L}_{\hat{m}}, \text{ and } \mathcal{L}_{\hat{s}} \not\subseteq \mathcal{L}_0^\lambda, \quad \mathcal{L}_0^\lambda \not\subseteq \mathcal{L}_{\hat{s}}$$

Proof. Since the inclusions in the first two pairs have been shown before in Lemma 4 and Lemma 5, we have to verify their strictness. To do this, we define a labeled Petri net, K_* , see Fig. 2, for which $L_{\hat{s}}(K_*) = L_{\hat{m}}(K_*) = (D_1\{\$\}^*)^*$, if the initial and final marking is r . In K_* the places p and q , together, have exactly one token. Since the final marking is r , and there is no arc leaving the failure place q , the only interesting maximal step (multi-step, resp.) is the one with label $\$$. If it fires, and p is marked, then the final marking will never be reached. In order to return to the initial marking r , the maximal $\$$ -step (multi-step, resp.) can be fired only, if p is unmarked and then the transitions for x and $\bar{x}$ spell out some word from the semi-Dyck set D_1 .

Now, if $(D_1\{\$\}^*)^*$ would be contained in $\mathcal{L}_s$ or in $\mathcal{L}_m$ then it would also be contained in $\mathcal{M}_{\cap}(D_1) = \mathcal{L}_0^\lambda \supseteq \mathcal{L}_m \supseteq \mathcal{L}_s = \mathcal{L}_0$. However, then $(D_1\{\$\}^*)^* = (D_1\{\$\}^*)^* \cap (\{x, \bar{x}\}^*\{\$\})^*$ were contained in $\mathcal{L}_0^\lambda$, but the least intersection-closed

Fig. 2. Petri net K_*

full trio containing $(D_1\{\$\})^*$ equals the family of recursively enumerable sets $\mathcal{Re}$, as follows from AFL-theory and is reported in [Grei 78, Jant 79a]. This then, would contradict the strict inclusion $\hat{\mathcal{M}}_\cap(D_1) \subsetneq \mathcal{Re}$ that follows from the decidability of the reachability problem, see [Mayr 81, Mayr 84, Kosa 82, Lamb 92]. From this one also gets $\mathcal{L}_s \not\subseteq \mathcal{L}_0^\lambda$ and $\mathcal{L}_{\hat{m}} \not\subseteq \mathcal{L}_0^\lambda$. Furthermore, the proof of Theorem 2 shows that the language L_{bin} is not an element of $\mathcal{L}_s$, while it is known (see [Jant 79a, Grei 78]) to be an element of $\mathcal{L}_0^\lambda$ (after having applied some trio operations to the similar language used there). $\square$

It is not known, whether it is possible to simulate λ -transitions in arbitrary Petri nets by using λ -free, maximal (multi-)steps. It is, however, possible to simulate a single multi-step by using λ -transitions:

Theorem 3.

$$\mathcal{L}_m^\lambda \subseteq \mathcal{L}_0^\lambda$$

Proof. Let $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$ be a labeled Petri net, where $N = (P, T, \partial_0, \partial_1)$. Then define

$$\begin{aligned} P' &:= (P \times \{0, 1\}) \cup \{p_x, p'_x \mid x \in \Sigma^\lambda\} \cup \{q\}, \\ T' &:= \{t, t', t'' \mid t \in T\} \cup \{t_x, t'_x \mid x \in \Sigma^\lambda\} \cup \{t_p \mid p \in P\}, \end{aligned}$$

where q and the p_x, p'_x ($x \in \Sigma^\lambda$) are new places and the t_x, t'_x ($x \in \Sigma^\lambda$) and t_p ($p \in P$) and t', t'' ($t \in T$) are new transitions. In the new net, in every reachable marking precisely one place in $S := \{q\} \cup \{p_x, p'_x \mid x \in \Sigma^\lambda\}$ will contain exactly one token. Therefore, the place in this set containing a token is called the *state*. For the mappings $\iota_i : P \rightarrow P \times \{0, 1\}$, $p \mapsto (p, i)$, $i = 0, 1$, we now consider the induced homomorphisms $\iota_i^* : P^\oplus \rightarrow (P \times \{0, 1\})^\oplus$. With these, we define

$$\partial'_i(t) := \iota_i^*(\partial_i(t)) \oplus p_{\sigma(t)}, \quad \partial'_i(t'') := \iota_i^*(\partial_i(t)) \oplus p'_{\sigma(t)}, \quad i = 0, 1,$$

$$\partial'_0(t') := \iota_0^*(\partial_0(t)) \oplus p_{\sigma(t)},$$

$$\partial'_1(t') := \iota_1^*(\partial_1(t)) \oplus p'_{\sigma(t)},$$

for all $t \in T$. It is easily seen that in the new net, the transitions t, t', t'' ($t \in T$) work similar to t from the old net. Thereby, the pre-multiset of t is taken from $P \times \{0\}$ and the post-multiset is put to $P \times \{1\}$. Note further, that each such firing requires the presence of a token on the place $p_{\sigma(t)}$ or $p'_{\sigma(t)}$ and that the difference between t, t', t'' lies in the treatment of the state: t fires in state $p_{\sigma(t)}$, t' changes the state from $p_{\sigma(t)}$ to $p'_{\sigma(t)}$ and t'' fires in state $p'_{\sigma(t)}$. So if the place p_x holds a token, a sequence of firings of such transitions t, t', t'' with $\sigma(t) = x$ in the new net simulates the parallel firing of the corresponding transitions in the old net. It is also clear, that the state p'_x in the new net can be reached iff the multi-step in the old net contained at least one transition.

In the states p_x and p'_x , a parallel firing of the old net is simulated, as explained above. If there is a token on q , there are two possible actions. On the one hand, the transitions t_p ($p \in P$) can be used to transfer tokens from $P \times \{1\}$ back to $P \times \{0\}$:

$$\partial'_i(t_p) := (p, 1 - i) \oplus q, \quad i = 0, 1,$$

for all $p \in P$. On the other hand, a state p_x ($x \in \Sigma^\lambda$) can be entered using the transition t_x and p'_x can be left using t'_x :

$$\begin{aligned} \partial'_0(t_x) &:= q, \quad \partial'_1(t_x) := p_x, \\ \partial'_0(t'_x) &:= p'_x, \quad \partial'_1(t'_x) := q. \end{aligned}$$

Thus, it is guaranteed that the state q can only be reentered, if the simulated multi-step from the old net contained at least one transition.

When one of the states p_x is reached, the transition used to enter it carries the symbol of the simulated multi-step:

$$\sigma'(t_x) := x \quad \text{for every } x \in \Sigma^\lambda.$$

For every transition $t \in T' \setminus \{t_x \mid x \in \Sigma^\lambda\}$, let $\sigma'(t) := \lambda$. To complete the construction, let $m'_0 := \iota_0^*(m_0) \oplus q$ and $\mathcal{F}' := \{\iota_0^*(m) \oplus q \mid m \in \mathcal{F}\}$ and $K' = (\Sigma, N', \sigma', m'_0, \mathcal{F}')$, where $N' = (P', T', \partial'_0, \partial'_1)$. $\square$

Combining Theorem 3 with Lemma 1 and the known characterization of $\mathcal{L}_0^\lambda = \hat{\mathcal{M}}_\cap(D_1)$ (see [Grei 78, Jant 79a]) we get:

Corollary 1.

$$\hat{\mathcal{M}}_\cap(D_1) = \mathcal{L}_0^\lambda = \mathcal{L}_s^\lambda = \mathcal{L}_m^\lambda$$

From this we already deduce $\mathcal{L}_m \wedge \mathcal{L}_m := \{L_1 \cap L_2 \mid L_1, L_2 \in \mathcal{L}_m\} \subseteq \mathcal{L}_0^\lambda$, which describes direct synchronization of different Petri nets using interleaving semantics. Moreover, we can show that equally labeled multi-steps in different Petri nets can be synchronized in a hand-shake manner:

Theorem 4. *The families $\mathcal{L}_{\hat{s}}$ and $\mathcal{L}_{\hat{m}}$ are both closed with respect to intersection, that is:*

$$\mathcal{L}_{\hat{s}} \wedge \mathcal{L}_{\hat{s}} = \mathcal{L}_{\hat{s}} \quad \text{and} \quad \mathcal{L}_{\hat{m}} \wedge \mathcal{L}_{\hat{m}} = \mathcal{L}_{\hat{m}}$$

Proof. Since $\Sigma^* \in \mathcal{L}_{\hat{s}}$ implies $\mathcal{L}_{\hat{s}} \subseteq \mathcal{L}_{\hat{s}} \wedge \mathcal{L}_{\hat{s}}$, we only show $\mathcal{L}_{\hat{s}} \wedge \mathcal{L}_{\hat{s}} \subseteq \mathcal{L}_{\hat{s}}$ and $\mathcal{L}_{\hat{m}} \wedge \mathcal{L}_{\hat{m}} \subseteq \mathcal{L}_{\hat{m}}$. Let K_1, K_2 be two disjoint, labeled, Petri nets denoted by $K_i = (\Sigma, N_i, \sigma_i, m_{i,0}, \mathcal{F}_i)$ with $N_i = (P_i, T_i, \partial_{i,0}, \partial_{i,1})$ ($i \in \{1, 2\}$, $P_1 \cap P_2 = \emptyset$, and $T_1 \cap T_2 = \emptyset$).

The labeled Petri net K' with $L_{\hat{s}}(K') = L_{\hat{s}}(K_1) \cap L_{\hat{s}}(K_2)$ (as well as $L_{\hat{m}}(K') = L_{\hat{m}}(K_1) \cap L_{\hat{m}}(K_2)$) is defined as follows,

$K' = (\Sigma, N', \sigma', m'_0, \mathcal{F}')$ with $N' = (P', T', \partial'_0, \partial'_1)$ with components described below:

As in the proof of Lemma 5 we add a failure place q and two places r_1, r_2 , thus $P' := P \cup \{q, r_1, r_2\}$. The places r_1 and r_2 are marked through the firing of the transitions and will take care, that in each step (resp. multi-step) at least one transition of $\tilde{T}_1$ and one of $\tilde{T}_2$ (to be defined next) is used. These places are added to the initial and the final markings of the respective nets yielding $m'_0 := m_{1,0} \oplus r_1 \oplus m_{2,0} \oplus r_2$ and $\mathcal{F}' := \{m_1 \oplus r_1 \oplus m_2 \oplus r_2 \mid \forall i = 1, 2 : m_i \in \mathcal{F}_i\}$.

In addition to the transitions $T_1 \cup T_2$, T' contains transitions $t_{i,x}, x \in \Sigma^\lambda, i \in \{1, 2\}$, labeled $\sigma'(t_{i,x}) := x$ to T_i , which may transfer the token from r_i to q , that is, $\forall i \in \{1, 2\} : \forall x \in \Sigma^\lambda : \partial'_0(t_{i,x}) := r_i, \partial'_1(t_{i,x}) := q$. This takes care, that any maximal step (leading to a final marking) avoids these transitions and has to use at least one of those in $\tilde{T} := \tilde{T}_1 \cup \tilde{T}_2$, defined next, and other transitions from the underlying nets N_1, N_2 .

For each $i = 1, 2$ let $\tilde{T}_i := \{t' \mid t \in T_i\}$, with $\partial'_0(t') := r_i \oplus \partial_{i,0}(t)$, $\partial'_1(t') := r_i \oplus \partial_1(t)$, and $\sigma'(t') := \sigma(t)$.

Any reachable marking of the new Petri net K' is of the form $m_1 \oplus r_1 \oplus m_2 \oplus r_2$, where m_i is a reachable marking in K_i , $i = 1, 2$.

The initial marking consists of the initial markings of old nets and in any maximal step (resp. maximal multi-step) exactly one transition from $\tilde{T}_1$ and exactly one from $\tilde{T}_2$ has to be used. The effect of this and any other transition with the same label is the same as if there had been multi-steps in the components m_1 and m_2 in the old nets. Hence, even without a more formal proof, the statement of the theorem follows. $\square$

When we compare the classes $\mathcal{L}_m$ and $\mathcal{L}_{\hat{m}}$ with families from the Chomsky hierarchy, especially with $\mathcal{C}_s = \text{NSpace}(n)$, we observe a similar result as in the investigation of the CFA-Languages from [JaKZ 07a] and [FJKRZ 07]:

Theorem 5.

$$\mathcal{L}_m \subsetneq \mathcal{C}_s \quad \text{and} \quad \mathcal{L}_{\hat{m}} \subsetneq \mathcal{C}_s$$

Proof. We will describe and analyse two algorithms: Algorithm 1, by which a multi-step language is accepted and its variation, Algorithm 2, by which the acceptance of the maximal multi-step language defined by a labeled Petri net $K = (\Sigma, N, \sigma, m_0, \mathcal{F})$, with $N = (P, T, \partial_0, \partial_1)$ is guaranteed.

Algorithm 1

BEGIN

1. $k := \max \left\{ \frac{|\partial_1(t)|}{|\partial_0(t)|} \mid t \in T \right\};$
2. Input $w = a_1 \cdots a_n \in \Sigma^+, a_1, \dots, a_n \in \Sigma;$
3. $m := m_0;$
4. FOR $i = 1, 2, \dots, n$ DO
5. Choose a multiset $\nu \in T_{a_i}^\oplus$ with $|\nu| \leq k^n |m_0|;$
6. IF $\partial_0(\nu) \not\sqsubseteq m$ THEN
7. Exit without accepting.
8. $m := m \ominus \partial_0(\nu) \oplus \partial_1(\nu);$
9. END FOR
10. IF $m \in \mathcal{F}$ THEN
11. Accept and exit.
12. ELSE
13. Exit without accepting.
14. END IF

END

Algorithm 2 is defined from Algorithm 1 by replacing the inner of the for-loop, i.e., lines 6. and 7. by the following:

Replace lines 6. and 7. in Algorithm 1 to get Algorithm 2 by:

6. IF $\partial_0(\nu) \sqsubseteq m$ THEN
- 7a. FOR ALL $t \in T$ DO
- 7b. IF $\partial_0(\nu \oplus t) \sqsubseteq m$ THEN (test of maximality)
- 7c. Exit without accepting. (ν is not maximal)
- 7d. END FOR
- 7e. ELSE
- 7f. Exit without accepting.
- 7g. END IF (ν is maximal for m)

We will verify that both algorithms work in $\text{NTimeSpace}(p(n), n)$, where p is a polynomial of low degree. Actually, these algorithms are only a minor modification of Algorithm 1 in [JaKZ 07b, JaKZ 08], where it was shown, that the languages accepted by Concurrent Finite Automata are in $\text{NTimeSpace}(n^2, n)$.

In the lines 4. to 9. of Algorithm 1, and of Algorithm 2, every cycle of the loop simulates one multi-step (maximal multi-step, resp.) in a computation

$$m_0 \xRightarrow[a_1]{\quad} m_1 \xRightarrow[a_2]{\quad} m_2 \cdots \xRightarrow[a_n]{\quad} m_n.$$

In each single multi-step $m_{i-1} \xRightarrow[a_i]{\quad} m_i, 1 \leq i \leq n$, we know that $|m_{i-1}| \leq k \cdot |m_i|$, hence $\forall i \leq n : |m_i| \leq k^i \cdot |m_0|$. Each marking m_i can be stored as a $|P|$ -tuple of binary encodings of $m_i(p)$.

Since $\forall p \in P : m_i(p) \leq k^i \cdot m_0(p) \leq k^n \cdot m_0(p)$ their binary encodings have a length that is bounded by an $O(n)$ function. Likewise, if $m_{i-1} \xRightarrow[a_i]{\quad} m_i, 1 \leq i \leq n$, with $m_i = m_{i-1} \ominus \partial_0(\nu_i) \oplus \partial_1(\nu_i)$ for some $\nu_i \in T_{a_i}^\oplus$, then $|\nu_i| \leq |m_{i-1}|$ since at most each single token in m_{i-1} is replaced using one transition from the multiset ν_i . Hence, $\forall i \leq n : |\nu_i| \leq k^n \cdot m_0$ and also these multisets can be stored in $\text{Space}(O(n))$. In Algorithm 1, the multisets chosen in line 5. are bounded by exactly this size, and the algorithm finds each multi-step sequence in linear space, and already $\mathcal{L}_m \subseteq \mathcal{C}_s$ follows. If we consider the time needed by Algorithm 1, we see $\mathcal{L}_m \subseteq \text{NTimeSpace}(n^2, n) \subseteq \mathcal{C}_s$: Each line in Algorithm 1 can be performed, non-deterministically because of line 5., within time $\text{NTime}(O(n))$, thus a single multi-step can be checked within $\text{NTime}(O(n))$ and the whole input of length n thereby in $\text{NTime}(O(n^2))$.

For Algorithm 2, the same space bound suffices. The newly replaced lines 6. to 7g. take again time in $\text{NTime}(O(n))$, since line 7b. is taken only the constant number $|T|$ of times, so that Algorithm 2 works in $\text{NTimeSpace}(n^2, n)$, too.

The proof for strict inclusion is based on the Time Hierarchy Theorem (Theorem 3.4.6 and Theorem 3.2.2 in [Reis 99], see the original papers [SeFM 78] and [Zak 83]), and will appear in the final journal publication [JaKZ 08] of [JaKZ 07b]. The proof in [JaKZ 07b] is only a reference to [Reis 99] which does not apply without a little more work. It is shown in [JaKZ 08], how one can adapt this result to our context and use it for proving $\text{NTimeSpace}(n^2, n) \subsetneq \mathcal{C}_s$. It follows, that in Theorem 5 actually strict inclusions $\mathcal{L}_m \subsetneq \mathcal{C}_s$ and $\mathcal{L}_{\hat{m}} \subsetneq \mathcal{C}_s$ are valid. $\square$

We can adapt the technique of Burkhard, [Burk 80, Burk 83], to prove universality of the maximum strategie in connection with λ -labeled transitions:

Theorem 6.

$$\mathcal{L}_s^\lambda = \mathcal{L}_{\hat{m}}^\lambda = \mathcal{R}_e.$$

Proof. We define a labeled Petri net K_A that simulates a computation of some non-deterministic counter automaton A by a λ -labeled, maximal step sequence, which – at the same time – is a maximal multi-step sequence:

For each counter c_i of the counter automaton, there will be a place p_{c_i} in K_A , and for each state q_i of the finite control of A , a place p_{q_i} . K_A will have p_{z_0} as initial marking, where z_0 is the initial state of the counter automaton, and $\emptyset$ as final marking. This can be achieved by adding λ -transitions to empty the places that correspond to final states of the finite control.

We use the following subnets as building blocks for a labeled Petri net:

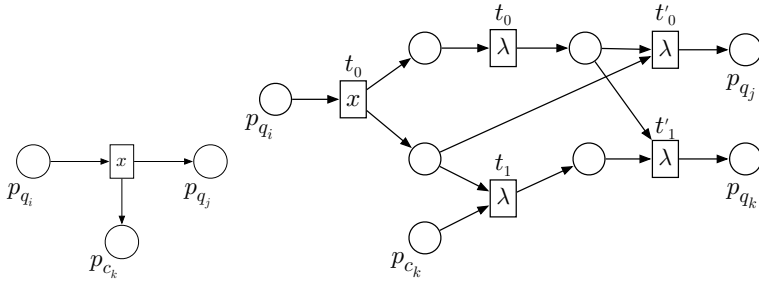


Fig. 3.

The smaller subnet on the left will be used for instructions of the type: From state q_i go to state q_j when reading x and add 1 to counter c_k . The other net can be used as encoding of the instruction: From state q_i go to state q_j when the counter c_k is 0 while reading x , else go to state q_k when reading x and subtract 1 from counter c_k . The labels of the transitions are written inside the boxes that represent the transitions. $\square$

4 Concluding Remarks

We study the effect of synchronized behaviour in Petri nets through steps and multi-steps. Thereby, we distinguish steps, as created by sets of parallel transitions from multi-steps, which can be seen as multiset sums of several steps. Only transitions having the same label from $\Sigma^\lambda = \Sigma \cup \{\lambda\}$ can form a (multi-)step. The label of a (multi-)step is used only once and the sequence of firable (multi-)steps defines the string used for the language created.

We compare various modes to fire transitions, including those with invisible (λ -labeled) transitions. Lemma 1 and Theorem 3 yield Corollary 1, showing that synchronization through labeled steps and multi-steps do not add more power to the sequential, unsynchronized firing of transitions, at least with respect to the languages definable this way. The discipline of maximality in firing (multi-)steps yields Turing equivalence, once invisible transitions are allowed (Theorem 6), which enriches the class of ordinary Petri net languages considerably. Without λ -transitions not more than context-sensitive languages are obtainable (Theorem 5).

From Lemma 3, Theorem 2, Theorem 6 and Lemma 7, we know incomparability of the families $\mathcal{L}_{\hat{s}}$ and $\mathcal{L}_s^\lambda$, and conjecture the same for the families $\mathcal{L}_{\hat{m}}$ and $\mathcal{L}_m^\lambda = \mathcal{L}_0^\lambda$.

Since $\mathcal{L}_{\hat{m}}$ and $\mathcal{L}_{\hat{s}}$ contain languages that are not context-free, the best we might get is the inclusion $\mathcal{Cf} \subsetneq \mathcal{L}_{\hat{m}}$. Yet, we conjecture incomparability of these two classes. The families $\mathcal{Cf}$ and $\mathcal{L}_{\hat{s}}$ have been proved to be incomparable in Lemma 6.

By adapting the technique in the proof of Theorem 3.2 in [JaKZ 08], it is possible to show $\mathcal{Cf} \subsetneq \mathcal{H}^{cod}(\mathcal{L}_{\hat{m}})$. However, we will use a more algebraic technique to prove this in an upcoming paper.

In this upcoming paper, we will also discuss in detail the distinction between the multi-step approach of this paper with that of the Petri net controlled finite automata from [JaKZ 08, FJKRZ 07]. In particular, we will compare the different usages of λ -transitions. In the firing mode of a CFA, the invisible λ -transitions are not synchronized at all, thus giving more and different possibilities than in the fully synchronized λ -labeled multi-steps.

Combining the results gained so far, we obtain the picture of Fig. 4.

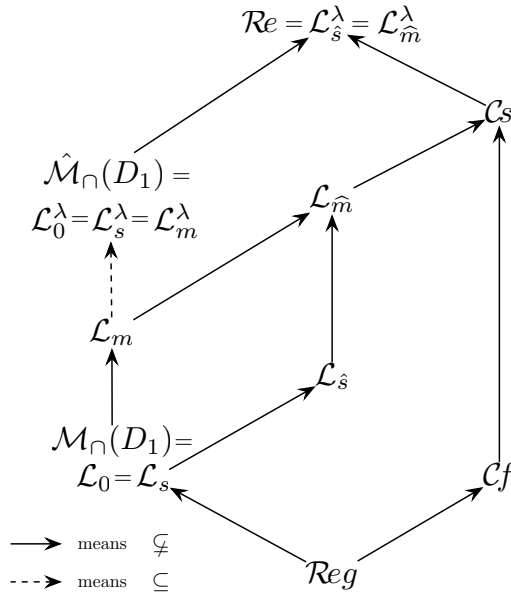


Fig. 4. Overview

References

- [Bake 72] Baker, H.: Petri nets and languages. C.S.G Memo 68, Project MAC. MIT, Cambridge (1972)
- [Burk 80] Burkhard, H.-D.: The Maximum Firing Strategie in Petri nets gives more power. ICS-PAS Report no. 411, Warschau, pp. 24–26 (1980)
- [Burk 81a] Burkhard, H.-D.: Ordered Firing in Petri Nets. EIK 17(2/3), 71–86 (1981)
- [Burk 81b] Burkhard, H.-D.: Two Pumping Lemmata for Petri Nets. EIK 17(2/3), 349–362 (1981)
- [Burk 83] Burkhard, H.-D.: On priorities of parallelism: Petri nets under the maximum firing strategy. In: Salwicki, A. (ed.) Logics of Programs and Their Applications. LNCS, vol. 148, pp. 86–97. Springer, Heidelberg (1983)

- [DiRo 95] Diekert, V., Rozenberg, G.: The book of traces. World Scientific Publishing, Singapore (1995)
- [FaKR 06] Farwer, B., Kudlek, M., Rölke, H.: Concurrent turing machines. *Fundamenta Informaticae* 79(3-4), 303–317 (2007)
- [FJKRZ 07] Farwer, B., Jantzen, M., Kudlek, M., Rölke, H., Zetsche, G.: On concurrent finite automata. In: Czaja, L. (ed.) *Proceedings of the workshop Concurrency, Specification and Programming (CS&P 2007)*, Łagów, Poland, September 2007, vol. 1, pp. 180–190 (2007)
- [Grei 78] Greibach, S.: Remarks on blind and partially blind one-way multcounter machines. *Theoretical Computer Science* 7, 311–324 (1978)
- [Hack 76] Hack, M.: *Petri Net Languages*, TR-159. MIT, Laboratory Computer Science, Cambridge, Mass (1976)
- [HaJa 94] Hauschildt, D., Jantzen, M.: Petri net algorithms in the theory of matrixgrammars. *Acta Informatica* 31, 719–728 (1994)
- [Jant 79a] Jantzen, M.: On the hierarchy of Petri net languages. *R.A.I.R.O., Informatique Théorique* 13, 19–30 (1979)
- [JaKZ 07a] Jantzen, M., Kudlek, M., Rölke, H., Zetsche, G.: Finite automata controlled by Petri nets. In: *Proceedings of the 14th workshop; Algorithmen und Werkzeuge für Petrinetze (S.Philippi, A.Pini eds.)* technical report Nr. 25/2007, Univ. Koblenz-Landau, pp. 57–62 (2007)
- [JaKZ 07b] Jantzen, M., Kudlek, M., Zetsche, G.: On languages accepted by concurrent finite automata. In: Czaja, L. (ed.) *Proceedings of the workshop Concurrency, Specification and Programming (CS&P 2007)*, Łagów, Poland, September 2007, vol. 2, pp. 321–332 (2007)
- [JaKZ 08] Jantzen, M., Kudlek, M., Zetsche, G.: Language classes defined by concurrent finite automata. *Fundamenta Informaticae* 85, 1–14 (2008)
- [Kosa 82] Kosaraju, S.R.: Decidability of reachability of vector addition systems. In: *14th Annual ACM Symp. on Theory of Computing*, San Francisco, pp. 267–281 (1982)
- [KuSa 86] Kuich, W., Salomaa, A.: Semirings, Automata, Languages. In: *EATCS-Monographs on Theoretical Computer Science*, vol. 5. Springer, Heidelberg, Berlin (1986)
- [Kudl 05] Kudlek, M.: Sequentiality, Parallelism, and Maximality in Petri Nets. In: Farwer, B., Moldt, D. (eds.) *Object Petri Nets, Processes, and Object Calculi*, pp. 43–50 (2005); techn. report FBI-HH-B-265/05
- [Lamb 92] Lambert, J.L.: A structure to decide reachability in Petri nets. *Theoretical Computer Science* 99, 79–104 (1992)
- [Mayr 81] Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 238–246 (1981)
- [Mayr 84] Mayr, E.W.: An algorithm for the general Petri net reachability problem. *SIAM J. of Computing* 13, 441–459 (1984)
- [Muku 92] Mukund, M.: Petri nets and step transition systems. *Internat. J. of Foundations of Computer Science* 3(4), 443–478 (1992)
- [NiPS 95] Nielsen, M., Priese, L., Sassone, V.: Characterizing Behavioural Congruences for Petri Nets. In: Lee, I., Smolka, S.A. (eds.) *CONCUR 1995*. LNCS, vol. 962, pp. 175–189. Springer, Heidelberg (1995)
- [Papa 94] Papadimitriou, C.H.: *Computational complexity*. Addison-Wesley, Reading (1949)
- [Pete 81] Peterson, J.L.: *Petri Nets and the Modelling of Systems*. MIT Press Series in Computer Science (1981)

- [Reis 99] Reischuk, K.R.: Komplexitätstheorie. In: Teubner, B.G. (ed.) Band I: Grundlagen, Stuttgart, Leipzig (1999)
- [SaMu 81] Salwicki, A., Müldner, T.: On the algorithmic properties of concurrent programs. In: Engeler, E. (ed.) Proceedings of the internat. Workshop: Logic of Programs, 1979, Zürich. LNCS, vol. 125, pp. 169–197. Springer, Heidelberg (1981)
- [SeFM 78] Seiferas, J., Fischer, M., Meyer, A.: Separating Nondeterministic Time Complexity Classes. *J. ACM* 25, 146–167 (1978)
- [Zak 83] Zak, S.: A TM Time Hierarchy. *Theoretical Computer Science* 26, 327–333 (1983)

MC-SOG: An LTL Model Checker Based on Symbolic Observation Graphs^{*}

Kais Klai¹ and Denis Poitrenaud²

¹ LIPN, CNRS UMR 7030
Université Paris 13
99 avenue Jean-Baptiste Clément
F-93430 Villetaneuse, France
`kais.klai@lipn.univ-paris13.fr`

² LIP6, CNRS UMR 7606
Université P. et M. Curie
104, avenue du Président Kennedy
75016 Paris, France
`Denis.Poitrenaud@lip6.fr`

Abstract. Model checking is a powerful and widespread technique for the verification of finite distributed systems. However, the main hindrance for wider application of this technique is the well-known state explosion problem. During the last two decades, numerous techniques have been proposed to cope with the state explosion problem in order to get a manageable state space. Among them, *on-the-fly* model-checking allows for generating only the "interesting" part of the model while *symbolic model-checking* aims at checking the property on a compact representation of the system by using Binary Decision Diagram (BDD) techniques. In this paper, we propose a technique which combines these two approaches to check $LTL \setminus X$ state-based properties over finite systems. During the model checking process, only an abstraction of the state space of the system, namely the *symbolic observation graph*, is (possibly partially) explored. The building of such an abstraction is guided by the property to be checked and is equivalent to the original state space graph of the system w.r.t. $LTL \setminus X$ logic (i.e. the abstraction satisfies a given formula φ iff the system satisfies φ). Our technique was implemented for systems modeled by Petri nets and compared to an explicit model-checker as well as to a symbolic one (NuSMV) and the obtained results are very competitive.

1 Introduction

Model checking is a powerful and widespread technique for the verification of finite distributed systems. Given a Linear-time Temporal Logic (LTL) property and a formal model of the system, it is usually based on converting the negation

^{*} The work presented in this paper is partially supported by the FME3 ANR Project: Enhancing the Evaluation of Error consequences using Formal Methods.

of the property in a Büchi automaton (or tableau), composing the automaton and the model, and finally checking for the emptiness of the synchronized product. The last step is the crucial stage of the verification process because of the state explosion problem. In fact, the number of reachable states of a distributed system grows exponentially with the number of its components. Numerous techniques have been proposed to cope with the state explosion problem during the last two decades. Among them the *symbolic model checking* (e.g. [6,10,12,4]) aims at checking the property on a compact representation of the system using binary decision diagrams (BDD) techniques [1], while the *On-the-fly model checking* (e.g., [13,8]) allows for generating only the "interesting" part of the state space. The exploration of the synchronized product is stopped as soon as the property is proved unsatisfied by the model. An execution scenario of the system illustrating the violation of the property (counter-example) can then be supplied in order to correct the model.

In this paper, we present a hybrid approach for checking linear time temporal logic properties of finite systems combining on-the-fly and symbolic approaches. Instead of composing the whole system with the Büchi automaton representing the negation of the formula to be checked, we propose to make the synchronization of the automaton with an abstraction of the original reachability graph of the system, namely a *state-based symbolic observation graph*. An event-based variant of the symbolic observation graph has already been introduced in [12]. Its construction is guided by the set of events occurring in the formula to be checked. Such events are said to be observed while the other events are unobserved. The event-based symbolic observation graph is then presented as a hybrid structure where nodes are sets of states (reachable by firing unobserved events) encoded symbolically and edges (corresponding to the firings of observed events) are represented explicitly. It supports on-the-fly model-checking and is equivalent to the reachability graph of the system with respect to event-based *LTL* semantic. Once built, the observation graph can be analyzed by any standard *LTL* $\setminus X$ model-checker. In [12], the evaluation of the method is only based on the size of the observation graph which is, in general, very moderate due to the small number of visible events occurring in a typical formula.

The event-based and state-based formalisms are interchangeable: an event can be encoded as a change in state variables, and likewise one can equip a state with different events to reflect different values of its internal variables. However, converting from one representation to the other often leads to a significant enlargement of the state space. Typically, event-based semantic is adopted to compare systems according to some equivalence or pre-order relation (e.g. [20,7,14]), while state-based semantic is more suitable to model-checking approaches [10,19].

The main contributions of this paper are, first the formal definition of a generalized state-based symbolic observation graph and the design and evaluation of an on-the-fly *LTL* $\setminus X$ (*LTL* minus the next operator) model-checker instantiating the approach for Petri net models.

The paper is structured as follows: In section 2, we introduce some preliminary definitions as well as some useful notations. In section 3, we formally define the

state-based symbolic observation graph and establish some preservation results. Then, section 4 describes the on-the-fly model checker tool implementation and gives a brief presentation of the way the *spot library* has been used for that purpose. The different algorithms were implemented in a software tool and experiments comparing our approach to both explicit on-the-fly and symbolic *LTL* model checkers are discussed in section 5. Finally, Section 6 concludes the paper and gives some perspectives.

2 Preliminaries

This section is dedicated to the definition of some relevant concepts and to the presentation of useful notations. The technique we present in this paper applies to different kinds of models, that can map to finite labeled transition systems, e.g. high-level bounded Petri nets. For the sake of simplicity and generality, we chose to present it for *Kripke structures*, since the formalism is rather simple and well adapted to state-based semantics.

Definition 1 (Kripke structure). *Let AP be a finite set of atomic propositions. A Kripke structure (KS for short) over AP is a 4-tuple $\langle \Gamma, L, \rightarrow, s_0 \rangle$ where:*

- Γ is a finite set of states ;
- $L : \Gamma \rightarrow 2^{AP}$ is a labeling (or interpretation) function;
- $\rightarrow \subseteq \Gamma \times \Gamma$ is a transition relation ;
- $s_0 \in \Gamma$ is the initial state.

Notations

- Let $s, s' \in \Gamma$. We denote by $s \rightarrow s'$ that $(s, s') \in \rightarrow$,
- Let $s \in \Gamma$. $s \not\rightarrow$ denotes that s is a *dead* state (i.e. $\nexists s' \in \Gamma$ such that $s \rightarrow s'$),
- $\pi = s_1 \rightarrow s_2 \rightarrow \dots$ is used to denote paths of a Kripke structure and $\overline{\pi}$ denotes the set of states occurring in π ,
- A finite path $\pi = s_1 \rightarrow \dots \rightarrow s_n$ is said to be a *circuit* if $s_n \rightarrow s_1$. If $\overline{\pi}$ is a subset of a set of states S then π is said to be a circuit of S .
- Let $\pi = s_1 \rightarrow \dots \rightarrow s_n$ and $\pi' = s_{n+1} \rightarrow \dots \rightarrow s_{n+m}$ be two paths such that $s_n \rightarrow s_{n+1}$. Then, $\pi\pi'$ denotes the path $s_1 \rightarrow \dots \rightarrow s_n \rightarrow s_{n+1} \rightarrow \dots \rightarrow s_{n+m}$.
- $\forall s, s' \in \Gamma$, $s \xrightarrow{*} s'$ denotes that s' is reachable from s (i.e. $\exists s_1, \dots, s_n \in \Gamma$ such that $s_1 \rightarrow \dots \rightarrow s_n \wedge s = s_1 \wedge s' = s_n$). $s \xrightarrow{+} s'$ denotes the case where $n > 1$ and $s \xrightarrow{*}_S s'$ (resp. $s \xrightarrow{+}_S s'$) stands when the states $s_1, \dots, s_n$ belong to some subset of states S .

Definition 2 (maximal paths). *Let $\mathcal{T}$ be Kripke structure and let $\pi = s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$ be a path of $\mathcal{T}$. Then, π is said to be a maximal path if one of the two following properties holds:*

- $s_n \not\rightarrow$,
- $\pi = s_1 \rightarrow \dots \rightarrow s_m \rightarrow \dots \rightarrow s_n$ and $s_m \rightarrow \dots \rightarrow s_n$ is a circuit.

Since LTL is interpreted on infinite paths, a usual solution in automata theoretic approach to check LTL formulae on a KS is to convert each of its finite maximal paths to an infinite one by adding a loop on its dead states. From now on, the KS obtained by such a transformation is called *extended Kripke structure* (EKS for short).

Moreover, it is well known that LTL formulae without the ‘next’ operator are invariant under the so-called *stuttering equivalence* [5]. We will use this equivalence relation to prove that observation graphs can be used for model checking. Stuttering occurs when the same atomic propositions (label) hold on two or more consecutive states of a given path. We recall the definition of *stuttering equivalence* between two paths.

Definition 3 (Stuttering equivalence). Let $\mathcal{T}$ and $\mathcal{T}'$ be two Kripke structures over an atomic proposition set AP and let $\pi = s_0 \rightarrow s_1 \rightarrow \dots$ and $\pi' = r_0 \rightarrow r_1 \rightarrow \dots$ be respectively paths of $\mathcal{T}$ and $\mathcal{T}'$. π and π' are said to be *stuttering equivalent*, written as $\pi \sim_{st} \pi'$, if there are two sequences of integers $i_0 = 0 < i_1 < i_2 < \dots$ and $j_0 = 0 < j_1 < j_2 < \dots$ s.t. for every $k \geq 0$, $L(s_{i_k}) = L(s_{i_{k+1}}) = \dots = L(s_{i_{k+1}-1}) = L'(r_{j_k}) = L'(r_{j_{k+1}}) = \dots = L'(r_{j_{k+1}-1})$.

3 Symbolic Observation Graph

Two motivations are behind the idea of the state-based *Symbolic Observation Graph* (SOG). First, state-based logics are more intuitive and more suitable for model-checking approaches than event-based ones. Second, we wanted to give a flexible definition of the SOG allowing several possible implementation which, as we will see in section 5, could improve the performances. In fact, thanks to the flexibility of the formal definition of the state-based SOG, the construction algorithm of [12] can be viewed as a specialization of our technique by observing events altering the truth values of the induced atomic propositions.

From now on, SOG will denote the state-based variant.

3.1 Definitions

We first define formally what is an *aggregate*.

Definition 4 (Aggregate). Let $\mathcal{T} = \langle \Gamma, L, \rightarrow, s_0 \rangle$ be a KS over an atomic proposition set AP . An aggregate a of $\mathcal{T}$ is a non empty subset of Γ satisfying $\forall s, s' \in a, L(s) = L(s')$.

We introduce three particular sets of states and two predicates. Let a and a' be two aggregates of $\mathcal{T}$.

- $Out(a) = \{s \in a \mid \exists s' \in \Gamma \setminus a, s \rightarrow s'\}$
- $Ext(a) = \{s' \in \Gamma \setminus a \mid \exists s \in a, s \rightarrow s'\}$

- $In(a, a') = \{s' \in a' \setminus a \mid \exists s \in a, s \rightarrow s'\}$ (i.e. $In(a, a') = Ext(a) \cap a'$)
- $Dead(a) = (\exists s \in a \text{ s.t. } s \nrightarrow)$
- $Live(a) = (\exists \pi \text{ a circuit of } a)$

Let us describe informally, for an aggregate a , the meaning of the above notations: $Out(a)$ denotes the set of *output states* of a i.e. any state of a having a successor outside of a . $Ext(a)$ contains any state, outside of a , having a predecessor in a . Given an aggregate a' , $In(a, a')$ denotes the set of *input states* of a' according to the predecessor a , notice that $In(a, a') = Ext(a) \cap a'$. Finally the predicate $Dead(a)$ (resp. $Live(a)$) holds when there exists a dead state (resp. a circuit) in a .

We first introduce the *compatibility* relation between an aggregate and a set of states.

Definition 5 (Compatibility). *An aggregate a is said compatible with a set of states S if and only if:*

- $S \subseteq a$
- $\forall s \in Out(a), \exists s' \in S$ such that $s' \xrightarrow{*}_a s$
- $Dead(a) \Rightarrow \exists s' \in S, \exists d \in a$ such that $d \not\rightarrow s' \xrightarrow{*}_a d$
- $Live(a) \Rightarrow \exists s' \in S, \exists \pi$ a circuit of a , $\exists c \in \pi$ such that $s' \xrightarrow{*}_a c$

Now, we are able to define the symbolic observation graph structure according to a given KS .

Definition 6 (Symbolic Observation Graph). *Let $\mathcal{T} = \langle \Gamma, L, \rightarrow, s_0 \rangle$ be a KS over an atomic proposition set AP . A symbolic observation graph of $\mathcal{T}$ is a 4-tuple $\mathcal{G} = \langle \Gamma', L', \rightarrow', a_0 \rangle$ where:*

1. $\Gamma' \subseteq 2^\Gamma$ is a finite set of aggregates
2. $L' : \Gamma' \rightarrow 2^{AP}$ is a labeling function satisfying $\forall a \in \Gamma'$, let $s \in a$, $L'(a) = L(s)$.
3. $\rightarrow' \subseteq \Gamma' \times \Gamma'$ is a transition relation satisfying:
 - (a) $\forall a, a' \in \Gamma'$ such that $a \rightarrow' a'$,
 - i. $a' \neq a \Rightarrow In(a, a') \neq \emptyset$ and a' is compatible with $In(a, a')$
 - ii. $a' = a \Rightarrow$ there exists a circuit π of a such that a is compatible with π and $\forall E \in In_{\mathcal{G}}(a)$, there exists a circuit π_E of a such that a is compatible with π_E and $\exists e \in E, c \in \pi_E$ satisfying $e \xrightarrow{*}_a c$ where $In_{\mathcal{G}}(a) = \{E \subseteq a \mid E = \{s_0\} \vee \exists a' \in \Gamma' \setminus \{a\}, a' \rightarrow' a \wedge E = In(a', a)\}$
 - (b) $\forall a \in \Gamma', Ext(a) = \bigcup_{a' \in \Gamma', a \rightarrow' a'} In(a, a')$.
4. $a_0 \in \Gamma'$ is the initial aggregate and is compatible with $\{s_0\}$

While points (1), (2) and (4) are trivial, the transition relation (3) of the *SOG* requires explanation: Given two aggregates a and a' such that $a \rightarrow' a'$, then we distinguish two cases:

(3(a)i) stands for a' is different from a . In this case, we impose a' to be compatible with $In(a, a')$. This means that entering in a' by following the arc

between a and a' , all the output states of a' , as well as a dead state (if $Dead(a')$ holds) and a circuit (if $Live(a')$ holds) must be reachable.

(3(a)ii) treats the non trivial case of a loop on a (i.e. $a \rightarrow' a$). Notice first that such a condition can be removed without changing the validity of our theoretical results. In this case, no loop will be authorized on a single aggregate a but it does not matter since $Live(a)$ holds and the stuttering implied by the loop will be captured by this predicate. However, this point appears to allow more flexibility for the *SOG* construction. A loop $a \rightarrow' a$ is authorized if for any subset E such that $E = \{s_0\}$ (if $s_0 \in a$) or $E = In(a', a)$ for some predecessor a' of a (in $\mathcal{G}$), there exists a cycle π_E reachable from E and a is compatible with $\overline{\pi_E}$. If such a subset E does not exist (i.e. $In_{\mathcal{G}}(a) = \emptyset$) then the aggregate a has no predecessor in $\mathcal{G}$ (and is not the initial aggregate) and we authorize a loop on a if and only if there exists a circuit π of a such that a is compatible with $\overline{\pi}$.

Finally, point (3b) implies that all the successors of the states present in an aggregate a are represented in Γ' .

Notice that Definition 6 does not guarantee the uniqueness of a *SOG* for a given *KS*. In fact, it supplies a certain flexibility for its implementation. In particular, an aggregate can be labeled by the same atomic proposition set than one of its successors and two successors of an aggregate may also have the same labels. We will see in the section 5 that the implementation can take advantage of this flexibility.

Example:

Figure 1 illustrates an example of *KS* (Figure 1(a)) and a corresponding *SOG* (Figure 1(b)). The set of atomic propositions contains two elements $\{a, b\}$ and each state of the *KS* is labeled with the values of these propositions. The presented *SOG* consists of 5 aggregates $\{a_0, a_1, a_2, a_3, a_4\}$ and 6 edges. Aggregates a_1 and a_2 contain circuits but no dead states, whereas a_3 and a_4 have each a dead state but no circuit. Each aggregate a is indexed with a triplet $(Dead(a), Live(a), L'(a))$. The symbol d (resp. $\bar{d}$) is used when $Dead(a)$ holds (resp. does not hold) and the symbol l (resp. $\bar{l}$) is used when $Live(a)$ holds (resp. does not hold). Notice that states of the *KS* are partitioned into aggregates which is not necessary the case in general (i.e. a single state may belong to two different aggregates). Moreover, one can merge a_3 and a_4 within a single aggregate and still respect Definition 6.

The following definition characterizes the paths of a *SOG* which must be considered for model checking.

Definition 7 (maximal paths of a *SOG*). *Let $\mathcal{G}$ be a *SOG* and $\pi = a_1 \rightarrow a_2 \rightarrow \dots \rightarrow a_n$ be a path of $\mathcal{G}$. Then π is said to be a maximal path if one of the three following properties holds:*

- $Dead(a_n)$ holds,
- $Live(a_n)$ holds,
- $\pi = a_1 \rightarrow \dots \rightarrow a_m \rightarrow \dots \rightarrow a_n$ and $a_m \rightarrow \dots \rightarrow a_n$ is a circuit (i.e. $a_n \rightarrow a_m$).

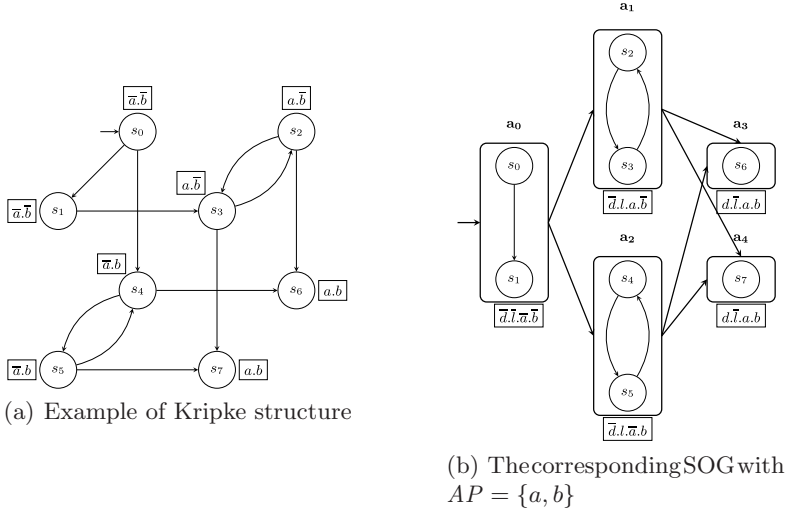


Fig. 1. A Kripke structure and its SOG

Our ultimate goal is to check $LTL \setminus X$ properties on a *SOG* $\mathcal{G}$ associated with a *KS* $\mathcal{T}$. Thus, in order to capture all the maximal paths of $\mathcal{G}$ under the form of infinite sequences, we transform its finite maximal paths into infinite ones. The following definition formalizes such a transformation and by analogy to extended Kripke structure, the obtained transformed graph is called *extended symbolic observation graph* (ESOG for short).

Definition 8 (Extended SOG). Let $\langle \Gamma', L', \rightarrow', a_0 \rangle$ be a SOG over an atomic proposition set AP . The associated ESOG is a KS $\langle \Gamma, L, \rightarrow, s_0 \rangle$ where:

1. $\Gamma = \Gamma' \cup \{v \in 2^{AP} \mid \exists a \in \Gamma', L'(a) = v \wedge (Dead(a) \vee Live(a))\}$
2. $\forall a \in \Gamma', L(a) = L'(a)$ and $\forall v \in \Gamma \setminus \Gamma', L(v) = v$
3. $\rightarrow \subseteq \Gamma \times \Gamma$ is the transition relation satisfying:
 - (a) $\forall a, a' \in \Gamma', a \rightarrow' a' \Rightarrow a \rightarrow a'$
 - (b) $\forall a \in \Gamma', Dead(a) \vee Live(a) \Rightarrow a \rightarrow L'(a)$
 - (c) $\forall v \in \Gamma \setminus \Gamma', v \rightarrow v$
4. $s_0 = s'_0$

Example:

The extended SOG of Figure 1(b) is obtained by adding three nodes $a, \overline{b}$, $\overline{a}, b$ (corresponding to a_1 and a_2 respectively because $Live(a_1)$ and $Live(a_2)$ hold and a, b (corresponding to both aggregates a_3 and a_4 because $Dead(a_3)$ and $Dead(a_4)$ hold). These three added states are labeled with $a, \overline{b}$, $\overline{a}, b$ and a, b respectively and have each a looping arc. We also add the following 4 arcs: one from a_1 to $a, \overline{b}$, one from a_2 to $\overline{a}, b$, one from a_3 to a, b and one from a_4 to a, b .

3.2 LTL\X Model Checking and SOG

The equivalence between checking a given $LTL \setminus X$ property on the observation graph and checking it on the original labeled transition system is ensured by the preservation of maximal paths. This corresponds to the CFFD semantic [14] which is exactly the weakest equivalence preserving next time-less linear temporal logic. Thus, the symbolic observation graph preserves the validity of formulae written in classical Manna-Pnueli linear time logic [15] (LTL) from which the “next operator” has been removed (because of the abstraction of the immediate successors) (see for instance [18,11]).

In the following, we give the main result of the paper: checking an $LTL \setminus X$ formula on a Kripke structure can be reduced to check it on a corresponding SOG. Due to the complexity of the proof of this result, it will be deduced from a set of intermediate lemmas and one proposition.

Theorem 1. *Let $\mathcal{G}$ be an ESOG over an atomic proposition set AP and corresponding to an extended KS $\mathcal{T}$. Let φ be a formula from $LTL \setminus X$ on AP . Then the following holds:*

$$\mathcal{T} \models \varphi \Leftrightarrow \mathcal{G} \models \varphi$$

Proof. The proof of Theorem 1 is direct from Proposition 1 (ensuring the preservation of maximal paths) and Definition 8. $\square$

Given a KS $\mathcal{T} = \langle I, L, \rightarrow, s_0 \rangle$ over an atomic proposition set AP and $\mathcal{G} = \langle I', L', \rightarrow', a_0 \rangle$ a SOG associated with $\mathcal{T}$ according to Definition 6, we present four lemmas about the correspondence between paths of $\mathcal{T}$ and those of $\mathcal{G}$. These lemmas are followed by Proposition 1. The first lemma demonstrates that each finite path in $\mathcal{T}$ has (at least) a corresponding path in $\mathcal{G}$.

Lemma 1. *Let $\pi = s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$ be a path of $\mathcal{T}$ and a_1 be an aggregate of $\mathcal{G}$ such that $s_1 \in a_1$. Then, there exists a path $a_1 \rightarrow' a_2 \rightarrow' \dots \rightarrow' a_m$ of $\mathcal{G}$ and a strictly increasing sequence of integers $i_1 = 1 < i_2 < \dots < i_{m+1} = n + 1$ satisfying $\{s_{i_k}, s_{i_k+1}, \dots, s_{i_{k+1}-1}\} \subseteq a_k$ for all $1 \leq k \leq m$.*

Proof. We proceed by induction on the length of π . If $n = 1$, knowing that $s_1 \in a_1$ concludes the proof. Let $n > 1$ and assume that $a_1 \rightarrow' a_2 \rightarrow' \dots \rightarrow' a_{m-1}$ and $i_1, \dots, i_m$ correspond to the terms of the lemma for the path $s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_{n-1}$. Then, $s_{n-1} \in a_{m-1}$. Let us distinguish two cases.

(i) If $s_n \in a_{m-1}$ then the path $a_1 \rightarrow' a_2 \rightarrow' \dots \rightarrow' a_{m-1}$ and the sequence $i_1, \dots, i_m + 1$ satisfy the proposition.

(ii) If $s_n \notin a_{m-1}$ then $s_n \in \text{Ext}(a_{m-1})$ and, by def. 6 (item 3b), there exists an aggregate a_m such that $a_{m-1} \rightarrow' a_m$ and $s_n \in \text{In}(a_{m-1}, a_m)$. As a consequence, the path $a_1 \rightarrow' a_2 \rightarrow' \dots \rightarrow' a_{m-1} \rightarrow' a_m$ and the sequence $i_1, \dots, i_m, i_m + 1$ satisfy the proposition. $\square$

The next lemma shows that the converse also holds.

Lemma 2. *Let $\pi = a_1 \rightarrow' a_2 \rightarrow' \dots \rightarrow' a_n$ be a path of $\mathcal{G}$. Then, there exists a path $e_1 \rightarrow (b_2 \xrightarrow{*}_{a_2} e_2) \rightarrow \dots \rightarrow b_n$ of $\mathcal{T}$ satisfying $e_1 \in a_1$ and $b_n \in a_n$.*

Proof. We consider π in reverse order and proceed by induction on its length. If $n = 1$, it is sufficient to choose a state $s_1 \in a_1$. If $n = 2$, we have to distinguish two cases.

(i) If $a_1 \neq a_2$ then, by def. 6 (item 3(a)i), $In(a_1, a_2) \neq \emptyset$ and, by definition of In , there exist $e_1 \in Out(a_1)$ and $b_2 \in In(a_1, a_2)$ such that $e_1 \rightarrow b_2$. This path verifies the proposition.

(ii) If $a_1 = a_2$ then, by def. 6 (item 3(a)ii), we know that there exists a circuit σ of a_1 . Let $b_2 \in \bar{\sigma}$. The path $b_2 \xrightarrow{+}_{a_1} b_2$ satisfies the proposition.

Let $n > 2$ and assume that $e_2 \rightarrow \dots \rightarrow b_n$ corresponds to the terms of the proposition for the path $a_2 \rightarrow' \dots \rightarrow' a_n$. We know that $e_2 \in a_2$. Here four cases have to be considered.

(iii) If $a_1 \neq a_2 \wedge e_2 \in Out(a_2)$ then, using def. 6 (item 3(a)i), we know that there exists a state $b_2 \in In(a_1, a_2)$ such that $b_2 \xrightarrow{*}_{a_2} e_2$ and a state $e_1 \in Out(a_1)$ such that $e_1 \rightarrow b_2$. The path $e_1 \rightarrow (b_2 \xrightarrow{*}_{a_2} e_2) \rightarrow \dots \rightarrow b_n$ verifies the proposition.

(iv) If $a_1 = a_2 \wedge e_2 \in Out(a_2)$ then, by def. 6 (item 3(a)ii), we know that a_1 contains a circuit σ such that a_1 is compatible with $\bar{\sigma}$. Let $e_1, b_2 \in \bar{\sigma}$ such that $e_1 \rightarrow b_2$. Since a_1 is compatible with $\bar{\sigma}$ and $e_2 \in Out(a_1)$ then e_2 is reachable from b_2 in a_1 . In consequence, the path $e_1 \rightarrow (b_2 \xrightarrow{*}_{a_2} e_2) \rightarrow \dots \rightarrow b_n$ satisfies the proposition.

(v) $a_1 \neq a_2 \wedge e_2 \notin Out(a_2)$ then $(a_2, a_2) \in \rightarrow'$. Since $In(a_1, a_2) \neq \emptyset$ and due to def. 6 (item 3(a)ii), there exists a circuit σ of a_2 reachable from some state $b_2 \in In(a_1, a_2)$ and such that a_2 is compatible with $\bar{\sigma}$. Moreover, there exists $e_1 \in Out(a_1)$ such that $e_1 \rightarrow b_2$. Let $c \in \bar{\sigma}$. Let us distinguish the two following subcases:

(a) If there exists $i > 2$ such that $e_i \in Out(a_i)$ then, let j be the smallest such an i . Then, since a_j is compatible with $\bar{\sigma}$, e_j is reachable in a_j from c . Hence, the path $e_1 \rightarrow b_2 \xrightarrow{*}_{a_2} c (\xrightarrow{+}_{a_2} c)^{j-1} \xrightarrow{*}_{a_j} e_j \dots \rightarrow b_n$ verifies the proposition.

(b) If for all $i > 2$, $e_i \notin Out(a_i)$ then the path $e_1 \rightarrow b_2 \xrightarrow{*}_{a_2} c (\xrightarrow{+}_{a_2} c)^{n-1}$ satisfies the proposition.

(vi) $a_1 = a_2 \wedge e_2 \notin Out(a_2)$ then, by def. 6 (item 3(a)ii), we know that a_1 contains a circuit σ such that a_1 is compatible with $\bar{\sigma}$. We also know that $e_2 \in \bar{\sigma}$ by construction. Let $e_1 \in \bar{\sigma}$ such that $e_1 \rightarrow e_2$. Then the path $e_1 \rightarrow (e_2 \xrightarrow{*}_{a_2} e_2) \rightarrow \dots \rightarrow b_n$ satisfies the proposition. $\square$

We are now in position to study the correspondence between maximal paths.

Lemma 3. *Let $\pi = s_0 \rightarrow \dots \rightarrow s_n$ be a maximal path of $\mathcal{T}$. Then, there exists a maximal path $\pi' = a_0 \rightarrow' \dots \rightarrow' a_m$ of $\mathcal{G}$ such that there exists a sequence of integers $i_0 = 0 < i_1 < \dots < i_{m+1} = n + 1$ satisfying $\{s_{i_k}, s_{i_k+1}, \dots, s_{i_{k+1}-1}\} \subseteq a_k$ for all $0 \leq k \leq m$.*

Proof. If s_n is a dead state then knowing that $s_0 \in a_0$ (Definition 6 (item 4) and using Lemma 1, we can construct a path $\pi' = a_0 \rightarrow a_2 \cdots a_m$ and the associated integer sequence corresponding to π . Because the last visited state of π belongs to a_m , $Dead(a_m)$ necessarily holds and π' is then a maximal path in the sense of Definition 7.

Now, if s_n is not a dead state then, one can decompose π as follows: $\pi = \pi_1 \pi_2$ s.t. $\pi_1 = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_{n-1}$ and $\pi_2 = s_n \rightarrow s_{n+1} \rightarrow \cdots \rightarrow s_{n+m}$ (where π_2 is a circuit). Once again, applying Lemma 1 from a_0 , one can construct a path $\pi'_1 = a_0 \rightarrow' a_1 \rightarrow' \cdots a_k$ corresponding to π_1 . The corresponding path of π'_2 can be also constructed applying the same lemma. However, this path must be constructed from a_k if $s_n \in a_k$ or from a successor of a_k containing s_n otherwise (Definition 6 ensures its existence). Let $\pi'_2 = a_{b_1} \rightarrow' a_{b_1+1} \rightarrow' \cdots a_{e_1}$ be this path.

Then, let us distinguish the following four cases:

1. if π'_2 is reduced to a single aggregate a then $\overline{\pi_2} \subseteq a$ and, because π_2 is a circuit of $\mathcal{T}$, $Live(a)$ holds. Then, the path $\pi'_1 \pi'_2$ is maximal in $\mathcal{G}$.
2. else if $a_{e_1} \rightarrow' a_{b_1} \wedge s_n \in In(a_{e_1}, a_{b_1})$ then π'_2 is a circuit of $\mathcal{G}$ and $\pi'_1 \pi'_2$ is a maximal path of $\mathcal{G}$ satisfying the proposition.
3. else if $s_n \in a_{e_1}$ (i.e. $a_{b_1} = a_{e_1}$) then the path $a_{b_1+1} \rightarrow' \cdots a_{e_2}$ is a circuit of $\mathcal{G}$ and $\pi'_1 \rightarrow' a_{b_1} \rightarrow' a_{b_1+1} \rightarrow' \cdots a_{e_2}$ is a maximal path of $\mathcal{G}$ satisfying the proposition.
4. else, by Definition 6, there exists a successor of a_{e_1} containing s_n . Applying again Lemma 1 from this aggregate, we can construct a new path in $\mathcal{G}$ corresponding to π_2 . Let $a_{b_2} \rightarrow' a_{b_2+1} \rightarrow' \cdots a_{e_2}$ be this path. If we can deduce a circuit of $\mathcal{G}$ from this path applying one of the three above points, this concludes the proof. Otherwise, it is also possible to construct a circuit of $\mathcal{G}$ by linking a_{e_2} to a_{b_1} similarly to the point 2 and 3 above and deduce a circuit. If this is not the case, we can construct a new path corresponding to π_2 starting from a successor of a_{e_2} . Because the number of aggregates in $\mathcal{G}$ is finite, a circuit will be necessarily obtained.

Notice that for all the previous cases above, a sequence of integers can be easily constructed from the ones produced by Lemma 1. $\square$

Lemma 4. Let $\pi' = a_0 \rightarrow' \cdots \rightarrow' a_n$ be a maximal path of $\mathcal{G}$. Then, there exists a maximal path $\pi = (s_0 \xrightarrow{*}_{a_0} e_0) \rightarrow \cdots \rightarrow (b_n \xrightarrow{*}_{a_n} e_n)$ of $\mathcal{T}$.

Proof. Let π' be a maximal path reaching an aggregate a_n such that $Dead(a_n)$ or $Live(a_n)$ hold. First, let us notice that the proof is trivial if the path π' is reduced to a single aggregate because of the compatibility of a_0 with $\{s_0\}$ which implies that a dead state (resp. a state of a circuit of a_0) is reachable from s_0 .

Otherwise, using Lemma 2, there exists a path $\pi = e_0 \rightarrow (b_1 \xrightarrow{*}_{a_1} e_1) \rightarrow \cdots \rightarrow b_n$ of $\mathcal{T}$ satisfying $e_0 \in a_0$ and $b_n \in a_n$. If $e_0 \in Out(a_0)$, we have $s_0 \xrightarrow{*}_{a_0} e_0$ since a_0 is compatible with $\{s_0\}$. Otherwise, e_0 belongs to a circuit of a_0 and there exists in $\mathcal{G}$ an arc from a_0 to itself. Definition 6 (item 3(a)ii) ensures that this circuit can be chosen to be reachable from s_0 (and compatible with a_0) during the

construction of π . Finally, there exists a state $e_n \in a_n$ such that $b_n \xrightarrow{*}_{a_n} e_n$, where e_n is a dead state (if $Dead(a_n)$ holds) or a state of a circuit of a_n (if $Live(a_n)$ holds), because a_n is compatible with $In(a_{n-1}, a_n)$. Thus, the path $(s_0 \xrightarrow{*}_{a_0} e_0) \rightarrow (b_1 \xrightarrow{*}_{a_1} e_1) \rightarrow \dots \rightarrow (b_n \xrightarrow{*}_{a_n} e_n)$ satisfies the lemma.

Now, if neither $Dead(a_n)$ nor $Live(a_n)$ hold, then by Definition 7, $\pi' = a_0 \rightarrow \dots \rightarrow a_m \rightarrow \dots \rightarrow a_n$ with $a_m \rightarrow \dots \rightarrow a_n$ a circuit of $\mathcal{G}$. We distinguish the two following cases:

1. If $\forall m \leq i \leq n, a_i = a_m$. Using Lemma 2, we can construct a path of $\mathcal{T}$, namely $\pi = e_0 \rightarrow (b_1 \xrightarrow{*}_{a_1} e_1) \rightarrow \dots \rightarrow b_m$ corresponding to $a_0 \rightarrow \dots \rightarrow a_m$ such that e_0 is chosen to be reachable from s_0 (similarly to the above case). Because $a_m \rightarrow a_m$, a_m contains a circuit and b_m can be chosen such that this circuit is reachable from b_m . This leads to the construction of a maximal path of $\mathcal{T}$.
2. Otherwise, m can be chosen such that $a_m \neq a_n$ and $a_n \rightarrow' a_m$. From this decomposition of π' , Lemma 2 can be used to construct a maximal path of $\mathcal{T}$ satisfying the current lemma.

□

The following proposition is a direct consequence of the two previous lemmas.

Proposition 1. *Let $\mathcal{G} = \langle \Gamma', L', \rightarrow', a_0 \rangle$ be a SOG over an atomic proposition set AP and corresponding to a KS $\mathcal{T} = \langle \Gamma, L, \rightarrow, s_0 \rangle$. Then the following holds:*

1. $\forall \pi = s_0 \rightarrow s_1 \rightarrow \dots$, a maximal path of $\mathcal{T}$, $\exists \pi' = a_0 \rightarrow' a_1 \rightarrow' \dots$, a maximal path of $\mathcal{G}$ s.t. $\pi \sim_{st} \pi'$.
2. $\forall \pi' = a_0 \rightarrow' a_1 \rightarrow' \dots$, a maximal path of $\mathcal{G}$, $\exists \pi = s_0 \rightarrow s_1 \rightarrow \dots$, a maximal path of $\mathcal{T}$ s.t. $\pi \sim_{st} \pi'$.

Proof. The proof of Proposition 1 is direct by considering Lemmas 3 and 4 as well as Definitions 3 and 4. □

4 Construction of a SOG Model-Checker for Petri Nets

This section presents the model-checker (named MC-SOG) we have implemented to verify state based LTL/X formulae on bounded Petri nets [17] (i.e. nets having a finite reachability graph). The implementation of MC-SOG is based on Spot [9], an object-oriented model checking library written in C++. Spot offers a set of building blocks to experiment with and develop your own model checker. It is based on the automata theoretic approach to LTL model checking. In this approach, the checking algorithm visits the synchronized product of the ω -automaton corresponding to the negation of the formula and of the one corresponding to the system. Spot proposes a module for the translation of an LTL formula into an ω -automaton but is not dedicated to a specific formalism for the representation of the system to be checked. Then, many of the algorithms are based on a set of three abstract classes representing ω -automata and which must

be specialized depending on your own needs. The first abstract class defines a state, the second allows to iterate on the successors of a given state and the last one represents the whole ω -automaton. In our context, we have derived these classes for implementing ESOG of bounded Petri nets. It is important to notice that the effective construction of the ESOG is driven by the emptiness check algorithm of Spot and therefore, will be managed on-the-fly.

Spot is freely distributed under the terms of the GNU GPL (`spot.lip6.fr`) and has been recently qualified as one of the best explicit LTL model checkers [19].

We assume that the bound of the considered net is known *a priori*. Each set of markings corresponding to an aggregate is represented by a BDD using the encoding presented by Pastor & al in [16]. For a k -bounded place, this encoding uses $\lceil \log_2(k) \rceil$ BDD variables. The representation of the transition relation is achieved using two sets of variables. The first one allows to specify the enabling condition on the current state while the second represents the effect of the firing (see [16] for a more precise description).

The atomic propositions that we accept are place names. In a given marking, an atomic proposition associated to a place is satisfied if the place contains at least a token.

The construction of aggregates depends on places appearing as atomic proposition in the checked formula. In the following, we denote by AP this subset of places. The construction starts from a set of initial markings M , all of them satisfying exactly the same atomic propositions. To compute the initial aggregate of the ESOG, we state $M = \{m_0\}$. Then, the complete set of markings corresponding to an aggregate is obtained by applying until saturation (i.e. no new states can be reached any more) the transition relation limited to the transitions which do not modify the truth value of atomic propositions. Instead of checking this constraint explicitly, we statically restrict the set of Petri net transitions to be considered to the ones which do not modify the marking of places used as atomic propositions (i.e. the set $Local = \{t \in T \mid \forall p \in AP, Pre(p, t) = Post(p, t)\}$). We denote by $Sat(M)$ the set of markings reachable from at least a marking of M by the firings of transitions in $Local$ only. In other terms, $Sat(M)$ is defined by induction as follows:

- $\forall m \in M, m \in Sat(M)$,
- $\forall m \in Sat(M), (\exists t \in Local \text{ s.t. } m[t > m'] \Rightarrow m' \in Sat(M)$ (where $m[t > m']$ denotes that t is firable from m and its firing leads to m').

The successors of an aggregate are obtained by considering one by one the transitions excluded during the computation of aggregates (of the set $Extern = T \setminus Local$). For each of these transitions, we compute the set of markings directly reachable by firing it from the markings composing the aggregate. Then, these reached markings are the seed for the construction of the successor aggregate. For a transition $t \in Extern$, we define $Out(M, t) = \{m \in Sat(M) \mid m[t > \}$ and $Ext(M, t) = \{m' \mid \exists m \in Sat(M) \text{ s.t. } m[t > m']\}$. Notice that the definition of the set $Extern$ and the fact that the construction is started from $\{m_0\}$ ensure

that $\forall t \in Extern, \forall m, m' \in Sat(Ext(M, t))$ and $\forall p \in AP$, we have $m(p) = m'(p)$ and therefore, that all the markings of an aggregate satisfy the same atomic propositions.

We also note $Out(M) = \bigcup_{t \in Extern} Out(M, t)$. Finally, we define two predicates: $Dead(M)$ holds if $Sat(M)$ contains at least a dead marking and $Live(M)$ if $Sat(M)$ contains at least a circuit.

Starting from $M = \{m_0\}$, we construct on-the-fly an ESOG where aggregates are $Sat(M)$ and successors of such a node are $Sat(Ext(M, t))$ for each t of $Extern$ (satisfying $Ext(M, t) \neq \emptyset$). However, two aggregates $Sat(M)$ and $Sat(M')$ can be merged (i.e. identify as equal in the hash table of the emptiness check) if $Out(M) = Out(M') \wedge Dead(M) = Dead(M') \wedge Live(M) = Live(M')$ without contradicting the definition 6. This situation is illustrated in fig.2.

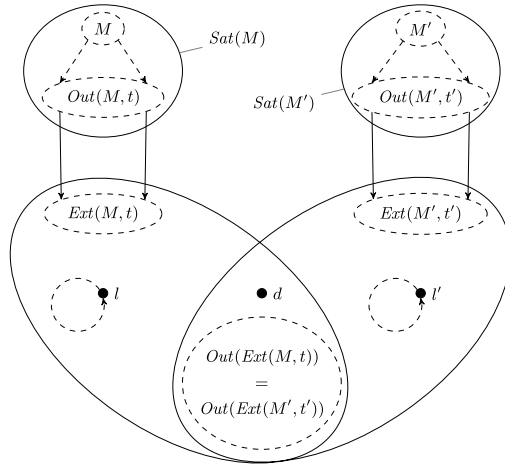


Fig. 2. The aggregates $Sat(Ext(M, t))$ and $Sat(Ext(M', t'))$ can be merged

This has an important consequence on the implementation. An aggregate, constructed from a set M , is only identified by the markings composing $Out(M)$ (one BDD) associated to the truth value of $Dead(M)$ and $Live(M)$ (two booleans). Then, the hash table is only composed by such triplets. This explains for an important part why the SOG construction obtains good results in terms of memory consumption. Indeed, when $Extern$ contains a limited number of transitions, the sets $Out(M)$ are generally small as well as the number of aggregates.

To determine if an aggregate contains a dead marking or a circuit, we use the algorithms presented in [12]. Moreover, when an aggregate contains one or the other, an artificial successor is added to the existing ones. This new aggregate is only characterized by the truth value of the atomic propositions (encoded by a BDD) and has itself as unique successor.

This computation is illustrated on the net of Figure 3 when considering two dining philosophers and the formula $\Box(wl_1 \wedge wr_1 \Rightarrow \Diamond(e_1))$. This formula

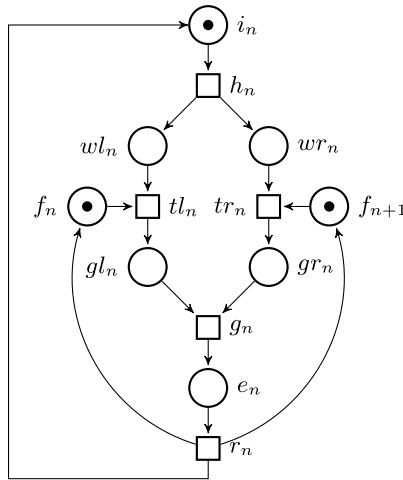


Fig. 3. A Petri net modeling the n^{th} dining philosopher

expresses that when the first philosopher wants to eat ($wl_1 \wedge wr_1$) then he must eat (e_1) eventually in the future. All the transitions connected to the three places wl_1 , wr_1 and e_1 are excluded during the computation of the markings composing an aggregate. Notice that these are all the transitions associated to the first philosopher. For instance, the initial aggregate, presented at the left of Figure 4, groups all the markings reachable from the initial marking when only the second philosopher runs. The transition h_1 is enabled from all these markings. Because this transition has been excluded during the computation and no other transition of *Extern* is enabled from any of these markings, the initial aggregate has only one successor whose construction begins with all the markings immediately reached after the firing of h_1 .

Notice that the initial aggregate contains no deadlock (d is false) but a circuit (l is true). Moreover, no atomic proposition is satisfied (wl_1 , wr_1 and e_1 are negated). Notice that the two dead markings of the net are represented in the two aggregates in the middle of the Figure 4. The artificial aggregates corresponding to the presence of dead markings or circuits have not been represented in the figure.

5 Evaluation

The performance of three LTL model checkers are compared. The first one, NuSMV [3], allows for the representation of synchronous and asynchronous finite state systems, and for the analysis of specifications expressed in Computation Tree Logic (CTL) and Linear Temporal Logic (LTL), using BDD-based and SAT-based model checking techniques. Only BDD-based LTL components have been used for our experiments. They implement the algorithms presented in [4]. This method consists in: (1) construct the transition relation of the system ;

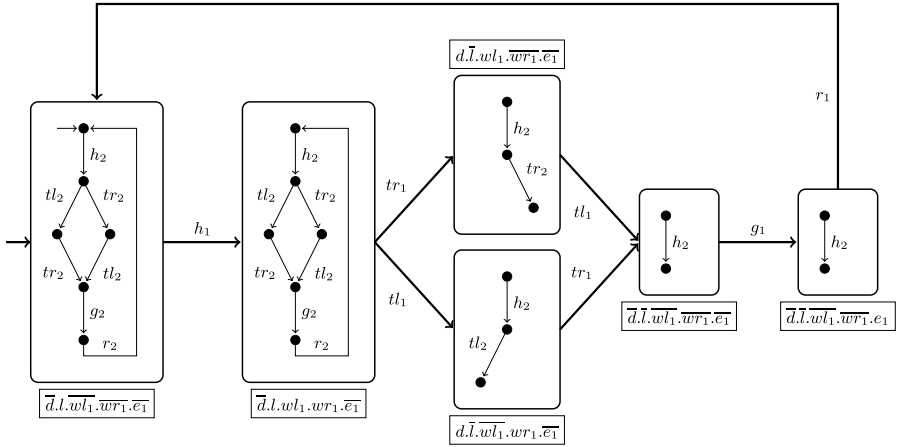


Fig. 4. The constructed SOG for two philosophers and the formula $\Box(wl_1 \wedge wr_1 \Rightarrow \Diamond(e_1))$

(2) translate the LTL formula into an equivalent ω -automaton and construct its transition relation ; (3) construct the synchronized product of the two relations. The decision procedure is then reduced to the verification of a CTL formula with fairness constraints. For our experiments, we have submitted to NuSMV, the encoding of Petri nets by the transition relation as defined by Pastor & al in [16]. On each dead marking, a transition looping on itself has been added to take into account all the maximal sequences.

The second model checker is MC-SOG presented in the previous section and mixing symbolic and explicit algorithms. Notice that the BDD representations of the transition relations used by this tool and NuSMV as well as the order of BDD variables are the same. Notice also that better encodings of considered nets may exist. In particular, the distribution of NuSMV proposes an example for the dining philosophers. This model encodes each philosopher and each fork by an automaton. This allows to significantly decrease the number of BDD variables comparing to the encoding we have chosen. However, a better encoding will be favorable for both tools (MC-SOG and NuSMV). In MC-SOG, the construction of aggregates is realized using this transition relation and the LTL verification is delegated to Spot [9].

The third model checker is also based on Spot (it is distributed with the library as a tutorial under the name CheckPN) but visits the reachability graph instead of a symbolic observation graph. Each state of the reachability graph is encoded by an integer vector. For this model checker also, we have added a transition looping on each dead marking.

The measurements presented in Table 1 concern 4 families of Petri nets. The Petri nets of a given family are obtained by instantiating a parameter (e.g. the number of philosophers). All these examples are taken from [2]. On each of these nets, 100 randomized formulae have been checked. Each formula is of size 8 and takes its atomic propositions randomly among the places of the considered net.

Table 1. Experiments on 4 families of Petri nets

model			Symbolic		Symbolic & Explicit				Explicit		
	(1)	(2)	(3)	(4)	(3)	(5)	(6)	(4)	(5)	(6)	(4)
fms 2	3444	26	801536	4.0	23207	762	1249	0.9	15522	50037	0.2
fms 3	48590	24	803790	9.2	23328	1359	2897	6.8	92942	346783	1.8
fms 4	438600	24	849616	25.7	51282	2617	7074	93.2	641853	2665186	15.2
fms 5	2.8×10^6	24	863802	48.6	91254	4917	15652	677.7	3391134	14956669	98.5
kanban 2	4600	31	772640	2.9	16715	871	1450	0.4	19014	108284	0.4
kanban 3	58400	30	770249	6.3	19185	1473	3183	0.7	196986	1437346	5.8
kanban 4	454475	30	783286	19.1	51451	2686	7121	2.9	1405537	11603892	328.0
kanban 5	2.5×10^6	30	810446	47.0	94913	4721	14317	7.5	not treated		
kanban 6	1.1×10^7	30	825585	124.4	178021	7911	26247	18.9			
philo 4	466	26	735463	4.2	11946	527	631	0.5	4709	12736	0.1
philo 6	10054	25	745489	11.6	18119	521	683	1.1	62250	293668	2.1
philo 8	216994	22	792613	32.3	24180	552	730	2.2	not treated		
philo 10	4.7×10^6	23	845180	98.7	30316	546	724	4.0			
philo 20	2.2×10^{13}	23	4743606	2585.5	59689	587	916	41.5			
ring 3	504	43	828719	10.8	11450	1621	4388	1.3	7541	24421	0.2
ring 4	5136	40	910641	66.6	18027	1922	4066	15.2	59145	295414	1.5
ring 5	53856	39	983840	438.0	69680	9682	29638	612.8	807737	4946514	181.2

(1) Number of reachable markings

(4) Verification time in seconds

(2) Number of verified formulae

(5) Number of visited states

(3) Peak number of live BDD nodes

(6) Number of visited transitions

(only an estimation for MC-SOG)

For each net, we give its number of reachable markings (column numbered (1)) as well as the number of satisfied formulae (2). For each tool, we have measured the time in seconds (4) consumed by the verification of the 100 formulae.

For each tool using the symbolic approach (NuSMV and MC-SOG), we also give the peak number of live BDD nodes (3) (i.e. peak even if aggressive garbage collection is used). Notice that the numbers given for MC-SOG are an estimation. Our implementation is based on the BDD package Buddy which delays the freeing of dead nodes to take advantage of its caches. Then, we have forced frequent garbage collections to estimate the peak. NuSMV is based on the BDD package CUDD which measures the exact memory peak in terms of live BDD nodes if only its immediate freeing function is used. NuSMV exclusively uses this function of CUDD.

For each tool using the explicit approach (MC-SOG and CheckPN) is indicated the number of states (5) and transitions (6) of the 100 ω -automata visited by the LTL emptiness check algorithm. These automata correspond to the synchronized products of the ESOG or reachability graph and the ω -automaton of each formula. Among all the emptiness check algorithms proposed by Spot, we have used the Couvreur's one [8]. The visited states are constructed (and stored) on-the-fly during the LTL verification. Then the number of states corresponds to the memory used and the number of transitions to the time consumed by the emptiness check.

A first remark concerning the obtained results in terms of time is that no model checker has the advantage on the others for all the experiments. NuSMV is the faster for the net **fms** and MC-SOG performs well for the nets **kanban** and **philo** while CheckPN obtains the best results for the net **ring**. The good results of NuSMV on **fms** are interesting. Only the number of tokens in the initial marking depends on the parameter of this net and, in particular, the number of places remains constant. Since the number of BDD variables is logarithmic with respect to the bound of the net when it is linear with respect to the number of places, we can deduce that NuSMV is more sensitive than MC-SOG to a great number of variables. We suspect that the bad results of MC-SOG for the net **fms** are essentially due to the way we construct aggregates and exclude some transitions for the saturation. This technique is particularly efficient when the bound of the places remains reasonable.

If we compare the memory peaks of NuSMV against the ones of MC-SOG, it is clear that NuSMV consumes more. This is due to the fact that MC-SOG only stores a forest of BDD (one by aggregate) corresponding to $Out(M)$ while NuSMV has to store the set of all reachable states. However, when the parameters of nets grow, the exponential increase of MC-SOG is more marked than the one of NuSMV (with the exception of the net **philo**).

Table 2. Experiments with limited expansion of aggregates

model			Symbolic		Symbolic & Explicit				Explicit		
	(1)	(2)	(3)	(4)	(3)	(5)	(6)	(4)	(5)	(6)	(4)
fms 5	2.8×10^6	24	863802	48.6	91254	10882	25989	52.1	3391134	14956669	98.5
fms 6	1.5×10^7	24	885576	147.6	293518	17104	45246	182.2	not treated		
ring 5	53856	39	983840	438.0	102668	56187	329075	239.3	807737	4946514	181.2

Finally, the size of the ω -automata visited by the explicit part (the emptiness checks) of MC-SOG is extremely reduced compared to the one required by CheckPN. It is clear that the computation time (which could be important) of MC-SOG is essentially used for the construction of the aggregates. However, we have seen in the section 3 that the definition of SOG allows some freedom degrees. In particular, the expansion of an aggregate can be stopped at any moment. Doing that, we limit the times required by the computation of aggregate but increase the size of the SOG. In table 2, we present some results for three nets. The expansion of aggregates has been limited by forbidding the firing of arbitrary chosen Petri net transitions during their construction (i.e. these transitions have been added to *Extern*). We can notice that the construction presented in the previous section does not allows the presence of self loop on aggregates. Adding some not needed transitions in *Extern* can lead to such loops but without contradicting the definition of SOG.

In table 2, we can remark that even if the size of visited ω -automata has been increased importantly, the complete checking time has decreased drastically and that, by using this simple heuristic, MC-SOG is now comparable with the two other model checkers on these examples.

6 Conclusion

In this paper, we designed and analyzed a new on-the-fly model-checker for $LTL \setminus X$ logic based on state-based symbolic observation graphs. Our approach is hybrid in the sense that the explored state space is represented by a hybrid graph: nodes are encoded symbolically while edges are encoded explicitly. In fact symbolic model-checker is rather common when dealing with computational tree logics (e.g. CTL), however checking LTL properties symbolically is not trivial. To the best of our knowledge the unique symbolic algorithm for LTL logic is the one proposed in [4] (implemented for instance in NuSMV [3]). The advantages of our technique in comparison to this approach is that the computation of a SOG can be done on-the-fly during the emptiness check. Moreover, when the SOG is visited entirely during the model checking (i.e. the property holds), it can be reused for the verification of another formula (at the condition that the set of atomic propositions is included in the one used by the SOG). Experiments show that our approach can outperform both explicit model-checkers and symbolic ones, especially when the freedom degrees related to the SOG building are exploited. However, it would be interesting to try different heuristics to limit more the computation time of aggregates. For instance, one can adapt the BDD variable ordering such that those used to encode the atomic propositions of the formula are consecutive. This could reduce drastically the time used for computing the successors of an aggregate since their values do not change within an aggregate. Other perspectives are to be considered in the near future. For example, it would be interesting to experiment our tool against realistic examples and not only toy ones.

References

1. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3), 293–318 (1992)
2. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Efficient symbolic state-space construction for asynchronous systems. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 103–122. Springer, Heidelberg (2000)
3. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer* 2(4), 410–425 (2000)
4. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. *Formal Methods in System Design* 10(1), 47–71 (1997)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: *Model Checking*. The MIT Press, Cambridge (2000)
6. Clarke, E.M., McMillan, K.L., Campos, S.V.A., Hartonas-Garmhausen, V.: Symbolic model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 419–427. Springer, Heidelberg (1996)
7. Cleaveland, R., Hennessy, M.: Testing equivalence as a bisimulation equivalence. In: *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pp. 11–23. Springer, New York (1990)

8. Couvreur, J.-M.: On-the-fly verification of linear temporal logic. In: Woodcock, J.C.P., Davies, J., Wing, J.M. (eds.) FM 1999. LNCS, vol. 1709, pp. 253–271. Springer, Heidelberg (1999)
9. Duret-Lutz, A., Poitrenaud, D.: SPOT: an extensible model checking library using transition-based generalized Büchi automata. In: Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), Volendam, The Netherlands, pp. 76–83. IEEE Computer Society Press, Los Alamitos (2004)
10. Geldenhuys, J., Valmari, A.: Techniques for smaller intermediary BDDs. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 233–247. Springer, Heidelberg (2001)
11. Goltz, U., Kuiper, R., Penczek, W.: Propositional temporal logics and equivalences. In: CONCUR, pp. 222–236 (1992)
12. Haddad, S., Ilić, J.-M., Klai, K.: Design and evaluation of a symbolic and abstraction-based model checker. In: Wang, F. (ed.) ATVA 2004. LNCS, vol. 3299, pp. 196–210. Springer, Heidelberg (2004)
13. Henzinger, T.A., Kupferman, O., Vardi, M.Y.: A space-efficient on-the-fly algorithm for real-time model checking. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 514–529. Springer, Heidelberg (1996)
14. Kaivola, R., Valmari, A.: The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 207–221. Springer, Heidelberg (1992)
15. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems. Springer, New York (1992)
16. Pastor, E., Roig, O., Cortadella, J., Badia, R.: Petri net analysis using boolean manipulation. In: Valette, R. (ed.) ICATPN 1994. LNCS, vol. 815, pp. 416–435. Springer, Heidelberg (1994)
17. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA (1981)
18. Puhakka, A., Valmari, A.: Weakest-congruence results for livelock-preserving equivalences. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 510–524. Springer, Heidelberg (1999)
19. Rozier, K., Vardi, M.: LTL satisfiability checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007)
20. Tao, Z.P., von Bochmann, G., Dssouli, R.: Verification and diagnosis of testing equivalence and reduction relation. In: ICNP 1995: Proceedings of the 1995 International Conference on Network Protocols, Washington, DC, USA, p. 14. IEEE Computer Society, Los Alamitos (1995)

Symbolic State Space of Stopwatch Petri Nets with Discrete-Time Semantics (Theory Paper)

Morgan Magnin, Didier Lime, and Olivier (H.) Roux

IRCCyN, CNRS UMR 6597, Nantes, France

{Morgan.Magnin,Didier Lime,Olivier-h.Roux}@irccyn.ec-nantes.fr

Abstract. In this paper, we address the class of bounded Petri nets with stopwatches (SwPNs), which is an extension of T-time Petri nets (TPNs) where time is associated with transitions. Contrary to TPNs, SwPNs encompass the notion of actions that can be reset, stopped and started. Models can be defined either with discrete-time or dense-time semantics. Unlike dense-time, discrete-time leads to combinatorial explosion (state space is computed by an exhaustive enumeration of states). We can however take advantage from discrete-time, especially when it comes to SwPNs: state and marking reachability problems, undecidable even for bounded nets, become decidable once discrete-time is considered. Thus, to mitigate the issue of combinatorial explosion, we now aim to extend the well-known symbolic handling of time (using convex polyhedra) to the discrete-time setting. This is basically done by computing the state space of discrete-time nets as the discretization of the state space of the corresponding dense-time model. First, we prove that this technique is correct for TPNs but not for SwPNs in general: in fact, for the latter, it may add behaviors that do not really belong to the evolution of the discrete-time net. To overcome this problem, we propose a splitting of the general polyhedron that encompasses the temporal information of the net into an union of simpler polyhedra which are safe with respect to the symbolic successor computation. We then give an algorithm that computes symbolically the state space of discrete-time SwPNs and finally exhibit a way to perform TCTL model-checking on this model.

Keywords: Verification using nets, Time Petri nets, symbolic state space, stopwatches, dense-time, discrete-time.

Introduction

The ever-growing development of embedded informatics requires efficient methods for the verification *a priori* of real-time systems. That is why researches on formalisms that allow engineers to write and check the interactions between CPUs and the communication networks appear as a hot topic. Time Petri Nets [1] are one of such formalisms. They model the temporal specifications of different actions under the form of time intervals. They can be enriched to represent tasks that may be suspended then resumed.

When modeling a system, either dense-time or discrete-time semantics may be considered. In the first one, time is considered as a dense quantity (i.e. the state of the system can change at any moment) and, in the second one, as a discrete variable (time progress is ensured by clock ticks and the global system may evolve only at these peculiar time steps). The physical systems (the processes) follow a dense-time evolution. The observation of the process is however usually performed through an IT command system which pilots it only at some peculiar instants (digitalization or periodic observations). In addition, the command system is composed of tasks that are executed on one (or many) processor(s) for which physical time is discrete. Dense-time is thus an over-approximation of the real system. The major advantage of dense-time lies in the symbolic abstractions it offers: they are easy to put into application and they avoid the combinatorial explosion of states. In this paper, we aim to propose a symbolic method to compute the state space of discrete-time Time Petri Nets (with stopwatches) by adapting the techniques usually dedicated to dense-time.

Time Petri Nets with Stopwatches

The two main time extensions of Petri nets are Time Petri nets [1] and Timed Petri nets [2]. In this paper, we focus on Time Petri nets (TPNs) in which transitions can be fired within a time interval.

In order to take into account the global complexity of systems, models now encompass the notion of actions that can be suspended and resumed. This implies extending traditional clock variables by "stopwatches". Several extensions of TPNs that address the modeling of stopwatches have been proposed: Scheduling-TPNs [3], Preemptive-TPNs [4] (these two models add resources and priorities attributes to the TPN formalism) and Inhibitor Hyperarc TPNs (ITPNs) [5]. ITPNs introduce special inhibitor arcs that control the progress of transitions. These three models belong to the class of TPNs extended with stopwatches (SwPNs) [6]. They have been studied in dense-time semantics.

In [6], state reachability for SwPNs has been proven undecidable, even when the net is bounded. As long as dense-time semantics is considered, the state space is generally infinite. Instead of enumerating each reachable state, verification algorithms compute finite abstractions of the state space, e.g. state class graph, that preserve the properties to be verified. But, as a consequence of the undecidability of the reachability problem, the finiteness of the state class graph cannot be guaranteed. In order to ensure termination on a subclass of bounded SwPNs, Berthomieu *et al.* propose an *overapproximation method* based on a quantization of the polyhedra representing temporal information [6]. Nevertheless the methods are quite costly in terms of computation time.

Relations between Discrete-Time and Dense-Time Semantics

In [7], Henzinger et al. compared the respective merits of dense-time and discrete-time approaches. They discussed the correctness of major model-checking problems according to the considered semantics. In other words, given a specification

Φ (that may be expressed correctly in both dense-time and discrete-time), what can be concluded for a system such that the discrete-time model satisfies Φ : does this mean Φ is also satisfied when dense-time semantics is considered? In this paper, we draw new links between dense-time and discrete-time semantics for models encompassing the notion of stopwatches.

In the case of SwPNs, the undecidability of major model-checking problems results from dense-time. The use of discrete-time instead (transitions are then no longer fired at any time but at integer dates) change these results, as we proved in [8]. In this paper, we established the following results:

- The state reachability problem - undecidable with dense-time semantics - is decidable when discrete-time is considered;
- The state space of discrete-time bounded SwPNs can be computed directly by using existing tools for classical Petri nets.

In the case of TPNs (without stopwatches), main works related to discrete-time are due to Popova. Her method consists in analyzing the behavior only at its so-called "integer-states" (i.e. states where the current local time for all enabled transitions are integers), which is sufficient to know the whole behavior of the net [9]. She defines a state as the conjunction of a marking and a time vector of the current local time for each of the enabled transitions and a special symbol for disabled transitions. Then she builds a reachability graph that is only based on the so-called "integer states". These are the states where the current local time of all enabled transitions are integers. She proves that the knowledge of the net behavior in these integer states is sufficient to determine the entire behavior of the net. This result, firstly proven for TPNs with finite latest firing times, has been later extended to nets with infinite latest firing times [10].

Following this work, Popova et al. presented, in [11], a parametric description of a firing sequence of a TPN that is based on the notion of integer states.

In the specific context of a comparison between dense and discrete-time, Popova's results establish the following assertion: a discrete-time analysis is sufficient to study a dense-time TPN. In this section, we propose the opposite result (for the state class graph, but our proof can easily be extended to zone graph), that is: the state space of a discrete-time TPN can be directly computed by discretizing its dense-time state space. In fact, this result does not only apply to TPNs.

In both cases, discrete-time based approaches suffer from a combinatorial explosion of the state space size. As efficient as the implementation (see [12] for data-structures dedicated to Petri nets and inspired by the Binary Decision Diagrams (BDDs)) could be, it reaches its limits as soon as there are transitions with a large time interval (e.g. [1, 10000]) in the model.

That is why we propose here to work out a method that would allow to compute symbolically the state space of discrete-time TPNs with stopwatches. The most natural idea consists in extending the method applied in dense-time to discrete-time, that means: compute the whole state space by the means of state classes that bring together all the equivalent discrete-time behaviors (the notion of equivalence will be precisely defined in section 3). Our approach consists

in computing the dense-time state-space *as long as it is significant* and then discretize it to get the discrete-time behavior.

State Space Computation of Dense-Time TPNs

For bounded dense-time TPNs, the state reachability problem is decidable. Although the state space of the model is infinite (as the clocks associated to transitions take their values in $\mathbb{R}$), it can be represented by a finite partition under the form of a state class graph [13] or a region graph [14]. The state class graph computes the whole set of reachable markings of a bounded TPN ; the resulting graph preserves the untimed language of the net (i.e. properties of linear temporal logics). However, it does not preserve branching temporal properties ; for this class of properties, we have to consider refinements of the state class graph, e.g. *atomic state classes* [15]. The zone graph can be used to check quantitative properties based on a subset of TPN-TCTL [16]. At the moment, only the first method has however been extended to TPNs with stopwatches [6]. Thus this is the approach we consider in this paper.

Our Contribution

In this paper, we address the general class of bounded Petri nets with stopwatches (SwPNs) with strict or weak temporal constraints and a single-server dense-time or discrete-time semantics. Our goal is to prove that, in certain cases, it is possible to compute the state space and the untimed language of a discrete-time net by simply discretizing the state space of the associated dense-time net. We exhibit an example showing however that, in the general case, this approach can lead to a false analysis. We conclude by giving a method that allies the efficiency of dense-time symbolic approaches with the specificities of discrete-time enumerative methods in order to compute a finite partition of the state space of TPNs with stopwatches. For the sake of simplicity, our results (based on mathematical considerations common to all TPNs extended with stopwatches models) are explained on a model whose high-level functions make them very convenient to understand: ITPNs.

Outline of the paper

Our aim is to compute efficiently the state space of a bounded discrete-time SwPN in order to verify quantitative timing properties. The paper is organized as follows: section 2 introduces TPNs with stopwatches (by the means of ITPNs) and the related semi-algorithm that computes their state space. In section 3, we show that, for classical TPNs, the discretization of the dense-time state space is sufficient to get the discrete-time state space. Section 4 extends the result to some subclasses of SwPNs but shows it is not valid for the general class of SwPNs. We exhibit a SwPN such that the discrete-time behaviors do not encompass all the dense-time behaviors in terms of untimed language and marking reachability.

In section 5, we propose a method, for computing the state space of SwPNs: this method combines the advantages of symbolic computations with dense-time specificities.

1 Time Petri Nets with Inhibitor Arcs

1.1 Notations

The sets $\mathbb{N}$, $\mathbb{Q}^+$ and $\mathbb{R}^+$ are respectively the sets of natural, non-negative rational and non-negative real numbers. An interval I of $\mathbb{R}^+$ is a $\mathbb{N}$ -interval iff its left endpoint belongs to $\mathbb{N}$ and its right endpoint belongs to $\mathbb{N} \cup \{\infty\}$. We set $I^\downarrow = \{x | x \leq y \text{ for some } y \in I\}$, the *downward closure* of I and $I^\uparrow = \{x | x \geq y \text{ for some } y \in I\}$, the *upward closure* of I . We denote by $\mathcal{I}(\mathbb{N})$ the set of $\mathbb{N}$ -intervals of $\mathbb{R}^+$.

1.2 Formal Definitions and Semantics of Time Petri Nets with Inhibitor Arcs

Definition 1. A *Time Petri net with Inhibitor Arcs (ITPN)* is a n -tuple $\mathcal{N} = (P, T, \bullet(\cdot), (\cdot)^\bullet, {}^\circ(\cdot), M_0, I)$, where

- $P = \{p_1, p_2, \dots, p_m\}$ is a non-empty finite set of places,
- $T = \{t_1, t_2, \dots, t_n\}$ is a non-empty finite set of transitions,
- $\bullet(\cdot) \in (\mathbb{N}^P)^T$ is the backward incidence function,
- $(\cdot)^\bullet \in (\mathbb{N}^P)^T$ is the forward incidence function,
- ${}^\circ(\cdot) \in (\mathbb{N}^P)^T$ is the inhibition function,
- $M_0 \in \mathbb{N}^P$ is the initial marking of the net,
- $I_s \in (\mathcal{I}(\mathbb{N}))^T$ is the function that associates a firing interval to each transition.

In [5], the authors extend inhibitor arcs with the notion of hyperarc. Inhibitor hyperarcs make it easier to model systems with priority relations between transitions, but they do not increase the theoretical expressivity of the model compared to inhibitor arcs. That is why we can equivalently work on Time Petri Nets with inhibitor arcs or inhibitor hyperarcs. For the sake of simplicity, we focus on nets with inhibitor arcs (ITPNs) in this paper.

A *marking* M of the net is an element of $\mathbb{N}^P$ such that $\forall p \in P, M(p)$ is the number of tokens in the place p .

A transition t is said to be *enabled* by the marking M if $M \geq \bullet t$, (i.e. if the number of tokens in M in each input place of t is greater or equal to the value on the arc between this place and the transition). We denote it by $t \in \text{enabled}(M)$.

A transition t is said to be *inhibited* by the marking M if the place connected to one of its inhibitor arc is marked with at least as many tokens than the weight of the considered inhibitor arc between this place and t : $0 < {}^\circ t \leq M$. We denote it by $t \in \text{inhibited}(M)$. Practically, inhibitor arcs are used to stop the elapsing of time for some transitions: an inhibitor arc between a place p and a transition t

means that the stopwatch associated to t is stopped as long as place p is marked with enough tokens.

A transition t is said to be *active* in the marking M if it is enabled and not inhibited by M .

A transition t is said to be *firable* when it has been enabled and not inhibited for at least $I(t)^\downarrow$ time units.

A transition t_k is said to be *newly* enabled by the firing of the transition t_i from the marking M , and we denote it by $\uparrow \text{enabled}(t_k, M, t_i)$, if the transition is enabled by the new marking $M - \bullet t_i + t_i^\bullet$ but was not by $M - \bullet t_i$, where M is the marking of the net before the firing of t_i . Formally:

$$\begin{aligned} \uparrow \text{enabled}(t_k, M, t_i) &= (\bullet t_k \leq M - \bullet t_i + t_i^\bullet) \\ \wedge ((t_k = t_i) \vee (\bullet t_k > M - \bullet t_i)) \end{aligned}$$

By extension, we will denote by $\uparrow \text{enabled}(M, t_i)$ the set of transitions newly enabled by firing the transition t_i from the marking M .

Let $\mathbb{T}$ be a generic time domain: it may be $\mathbb{N}$, $\mathbb{Q}^+$ or $\mathbb{R}^+$.

Definition 2. A state of a TPN is a pair $q = (M, I)$ in which M is a marking and I is a function called the interval function. Function $I \in (\mathcal{I}(\mathbb{N}))^T$ associates a temporal interval with every transition enabled at M .

We define the semantics of an ITPN as a time transition system. In this model, two kinds of transitions may occur: *time* transitions when time passes and *discrete* transitions when a transition of the net is fired.

Definition 3 (Semantics of an ITPN). Given a time domain $\mathbb{T}$, the semantics of a Time Petri Net with Inhibitor Arcs $\mathcal{N}$ is defined as a Timed Transition System $\mathcal{S}_{\mathcal{N}}^{\mathbb{T}} = (Q, q_0, \rightarrow)$ such that:

- $Q = \mathbb{N}^P \times \mathbb{T}^T$;
- $q_0 = (M_0, I_s)$
- $\rightarrow \in Q \times (T \cup \mathbb{T}) \times Q$ is the transition relation including a time transition relation and a discrete transition relation. The time transition relation is defined $\forall d \in \mathbb{T}$ by:

$$\begin{aligned} (M, I) &\xrightarrow{d} (M, I') \text{ iff } \forall t_i \in T, \\ \left\{ \begin{array}{l} I'(t_i) = \begin{cases} I(t_i) & \text{if } t_i \in \text{enabled}(M) \\ \text{and } t_i \in \text{inhibited}(M) \end{cases} \\ I'(t_i)^\uparrow = \max(0, I(t_i)^\uparrow - d), \text{ and } I'(t_i)^\downarrow = I(t_i)^\downarrow - d \text{ otherwise,} \\ M \geq \bullet t_i \Rightarrow I'(t_i)^\downarrow \geq 0 \end{array} \right. \end{aligned}$$

The discrete transition relation is defined $\forall t_i \in T$ by:

$$\begin{aligned} (M, I) &\xrightarrow{t_i} (M', I') \text{ iff } , \\ \left\{ \begin{array}{l} t_i \in \text{enabled}(M) \text{ and } t_i \notin \text{inhibited}(M), \\ M' = M - \bullet t_i + t_i^\bullet, \\ I(t_i) = 0, \\ \forall t_k \in T, I'(t_k) = \begin{cases} I_s(t_k) & \text{if } \uparrow \text{enabled}(t_k, M, t_i) \\ I(t_k) & \text{otherwise} \end{cases} \end{array} \right. \end{aligned}$$

In the *dense-time approach*, time is considered as a *continuous* variable whose evolution goes at rate 1. The dense-time semantics of the net $\mathcal{N}$ is thus $\mathcal{S}_{\mathcal{N}}^{dense} = \mathcal{S}_{\mathcal{N}}^{\mathbb{R}^+}$.

By contrast, in the *discrete-time approach*, time is seen as "jumping" from one integer to the other, with no care of what may happen in between. The latter is an under-approximation of the former. The discrete-time semantics of $\mathcal{N}$ is $\mathcal{S}_{\mathcal{N}}^{discrete} = \mathcal{S}_{\mathcal{N}}^{\mathbb{N}}$.

Note that $\mathcal{S}_{\mathcal{N}}^{discrete}$ can be straightforwardly reduced to a simple transition system. As long as discrete-time semantics is considered, any open interval with integer bounds may be turned into a closed interval. That is why, in the following, we only consider closed intervals (that may however be open on ∞).

Note also that for transitions which are not enabled, the time transition relation of the semantics lets the firing intervals evolve. They could as well have been stopped.

A run ρ of length $n \geq 0$ in a TTS is a finite or infinite sequence of alternating time and discrete transitions of the form

$$\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_0} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_1} \dots q_n \xrightarrow{d_n} \dots$$

We write $first(\rho)$ the first state of a run ρ . A run is *initial* if $first(\rho) = q_0$. A run ρ of $\mathcal{N}$ is an initial run of $\mathcal{S}_{\mathcal{N}}^{\mathbb{T}}$. The timed language accepted by $\mathcal{N}$ with dense-time semantics (respectively with discrete-time semantics) is $\mathcal{L}^{dense}(\mathcal{N}) = \mathcal{L}(\mathcal{S}_{\mathcal{N}}^{dense})$ (resp. $\mathcal{L}^{discrete}(\mathcal{N}) = \mathcal{L}(\mathcal{S}_{\mathcal{N}}^{discrete})$).

In the following, we denote by $\mathcal{Q}^{dense}$ (resp. $\mathcal{Q}^{discrete}$) the set of reachable states of $\mathcal{S}_{\mathcal{N}}^{dense}$ (resp. $\mathcal{S}_{\mathcal{N}}^{discrete}$).

To every TPN (possibly extended with stopwatches) structure $\mathcal{N}$, we can associate either a *dense-time* or a *discrete-time* semantics. We then obtain two different models. In the following, we say that two TPNs are *associated* if, whatever the choice on their semantics has been, they share the same underlying structure $\mathcal{N}$.

Definition 4. Given an integer $n \in \mathbb{N}$ and a set $D \subseteq \mathbb{R}^n$. We define the discretization operator of domain D by: $\mathcal{Disc}(D) = D \cap \mathbb{N}^n$

Definition 5. Given an integer $n \in \mathbb{N}$ and a set $D \subseteq \mathbb{R}^n$. A point $\theta = (\theta_1, \theta_2, \dots, \theta_n) \in D$ is an integer point of D if all its coordinates are integers, i.e. $\forall i, \theta_i \in \mathbb{N}$.

In this paper, we restrict ourselves to the class of *bounded* TPNs and ITPNs. This does not imply that the underlying net is bounded! The converse assertion is however true: the boundedness of the underlying PN ensures the boundedness of a TPN (or ITPN).

1.3 State Space Computation of Dense-Time Models

In order to analyze a Time Petri Net, the computation of its reachable state space is required. However, the reachable state space of a TPN is obviously infinite.

For models expressed with dense-time semantics, one of the approaches to partition the state space in a finite set of infinite state classes is the state class graph proposed by Berthomieu and Diaz [13]. It has been extended for TPNs with stopwatches [17].

A state class contains all the states of the net between the firing of two successive transitions.

Definition 6. A state class C of a dense-time ITPN is a pair (M, D) where M is a marking of the net and D a (convex) polyhedron with m constraints ($m \in \mathbb{N}$) involving up to n variables, with n being the number of transitions enabled by the marking of the class:

$$A\Theta \leq B$$

with A and B being rational matrices of respective dimensions (m, n) and $(m, 1)$ and $\Theta = (\theta_1, \theta_2, \dots, \theta_n)$ being a vector of dimension n (θ_i represents the firing time associated to transition t_i). D constrains the firing times θ_i of transitions.

In the case of TPNs, the firing domain is simpler than a general polyhedron: the authors of [18,19] have proved it can be encoded into the efficient Difference Bound Matrix (DBM) datastructure).

We extend the definition of the *discretization operator* of a point in $\mathbb{R}^n$ to state classes by the following definition:

Definition 7. Let $C = (M, D)$ be a state class of a TPN (with or without inhibitor arcs). We define the discretization operator of the state class C by : $Disc(C) = (M, Disc(D))$

In the case of ITPNs, the only fireable transitions are the active ones. So we need to define properly the firability of a transition from a class:

Definition 8 (Firability). Let $C = (M, D)$ be a state class of a ITPN. A transition t_i is said to be fireable from C iff there exists a set $(\theta_0^*, \dots, \theta_{n-1}^*)$ of D , such that $\forall j \in \{0, \dots, n-1\} - \{i\}$, s.t. t_j is active, $\theta_i^* \leq \theta_j^*$.

Now, given a class $C = (M, D)$ and a fireable transition t_f , the class $C' = (M', D')$ obtained from C by the firing of t_f is given by

- $M' = M - \bullet t_f + t_f^\bullet$
- D' is computed along the following steps, and noted $next(D, t_f)$
 1. intersection with the firability constraints : $\forall j$ s.t. t_j is active, $\theta_f \leq \theta_j$
 2. variable substitutions for all enabled transitions that are active t_j : $\theta_j = \theta_f + \theta'_j$,
 3. elimination (using for instance the Fourier-Motzkin method) of all variables relative to transitions disabled by the firing of t_f ,
 4. addition of inequations relative to newly enabled transitions

$$\forall t_k \in \uparrow enabled(M, t_f), I(t_k)^\downarrow \leq \theta'_k \leq I(t_k)^\uparrow$$

The variable substitutions correspond to a shift of time origin for active transitions: the new time origin is the firing time of t_f . t_f is supposed to be fireable so the polyhedron constrained by the inequalities $\theta_f \leq \theta_j$ is non empty.

The state class graph is generated by iteratively applying the function that computes the successors of a state class. The computation starts from the initial state class given by $C_0 = (M_0, D_0)$ with $D_0 = \{\theta_k \in I_s(t_k) \mid t_k \in \text{enabled}(M_0)\}$.

Definition 9 (Next). Let $C = (M, D)$ be a state class of a ITPN, let Θ be a point of D and t_f be a transition fireable from $(M, \{\Theta\})$. The successor $(M', \text{next}_{t_f}^{\text{dense}}(\{\Theta\}))$ of $(M, \{\Theta\})$ by firing t_f is given by $M' = M - \bullet t_f + t_f^\bullet$ and :

$$\text{next}_{t_f}^{\text{dense}}(\{\Theta\}) = \left\{ \forall i \in [1..n] [\theta'_1 \dots \theta'_n]^\top \left| \begin{array}{l} \theta'_i \in I_s(t_i) \text{ if } \uparrow \text{enabled}(t_i, M, t_f) \\ \theta'_i = \theta_i \text{ if } t_i \in \text{enabled}(M) \\ \text{and } t_i \in \text{inhibited}(M) \\ \text{and not } \uparrow \text{enabled}(t_i, M, t_f) \\ \theta'_i = \theta_i - \theta_f \text{ otherwise} \end{array} \right. \right\}$$

The $\text{next}_{t_f}^{\text{dense}}$ operator straightforwardly extends to finite or infinite unions of points.

1.4 State Space Computation of Discrete-Time Models

To our knowledge, the only existing methods for computing the state space of discrete-time TPNs (with or without stopwatches) are based on an enumeration of states. The computation does not necessarily finish (for example, if the net contains a transition with a latest firing time equal to infinity). It suffers from the combinatorial explosion of the state space size. A way to mitigate this issue consists in working with data-structured inspired from the well-known Binary Decision Diagrams (BDDs) [12]. This approach however reveals its limits when large timing constraints are implicated in the model.

In this paper, we propose a new way to deal with combinatorial explosions. It consists in extending the symbolic methods usually applied to dense-time to discrete-time. We thus define the notion of state classes in the specific context of discrete-time.

The underlying idea is the same as in dense-time: a state class contains all the states of the net between the firing of two successive transitions.

Definition 10. A state class C of a discrete-time ITPN is a pair (M, D) where M is a marking of the net and D a set (potentially empty) of points of $\mathbb{N}^n$, with n being the number of transitions enabled by the marking of the class.

The definition of firability remains the same in the discrete case.

Now, given a class $C = (M, D)$ and a fireable transition t_f , the successor class $C' = (M', D')$ obtained from C by the firing of t_f is denoted by $C' = (M', D') = (M', \text{next}_{t_f}^{\text{discrete}}(D))$ where $\text{next}_{t_f}^{\text{discrete}}$ is defined for all integer points Θ by

$$\text{next}_{t_f}^{\text{discrete}}(\{\Theta\}) = \text{Disc}(\text{next}_{t_f}^{\text{dense}}(\{\Theta\}))$$

Note that the operator $\mathcal{Disc}$ is necessary here because of the interval of the newly enabled transitions.

The purpose of this paper is to extend symbolic methods of dense-time to discrete-time according to the following approach:

- Describe the set of points of a temporal domain D not by an enumeration, but by a convex polyhedron $Poly$ such that $D = \mathcal{Disc}(Poly)$
- Compute $C^{symb'} = (M', Poly')$, successor of $C^{symb} = (M, Poly)$ by the firing of t_f (denoted $(M, next_{t_f}^{dense}(D))$) by the classical symbolic method and then link the symbolic classes $C^{symb}, C^{symb'}, \dots$ to the state space of $\mathcal{N}$ considered with its discrete-time semantics $\mathcal{S}_{\mathcal{N}}^{discrete}$

2 Relations between Dense-Time and Discrete-Time for Time Petri Nets with Stopwatches

2.1 Relations between Dense-Time and Discrete-Time Semantics in the Specific Case of Time Petri Nets

Theorem 1. *Let $\mathcal{N}$ be a TPN. Let C be one of the state classes of its dense-time semantics $\mathcal{S}_{\mathcal{N}}^{dense} C = (M, DBM)$. Let t_f be a firable transition from C . Then $next_{t_f}^{discrete}(\mathcal{Disc}(DBM)) = \mathcal{Disc}(next_{t_f}^{dense}(DBM))$*

Let us omit the proof of this theorem, since it actually is a special case of theorem 3, which will be proved in the next paragraph.

Theorem 1 leads to the following major and immediate corollary:

Corollary 1. *For all TPN $\mathcal{N}$, let $\mathcal{S}_{\mathcal{N}}^{dense} = (\mathcal{Q}^{dense}, q_0, \rightarrow^{dense})$ (resp. $\mathcal{S}_{\mathcal{N}}^{discrete} = (\mathcal{Q}^{discrete}, q_0, \rightarrow^{discrete})$) be its dense-time (resp. discrete-time) semantics. Then, $\mathcal{Q}^{discrete} = \mathcal{Disc}(\mathcal{Q}^{dense})$.*

2.2 Differences between Dense-Time and Discrete-Time Semantics in Terms of Marking Reachability and Untimed Language

We prove here that, in the general case of ITPNs, the state space of a discrete-time ITPN and the discretization of the state space of the associated dense-time net are not equal.

Theorem 2. *There exists a ITPN $\mathcal{N}$, with $\mathcal{S}_{\mathcal{N}}^{dense} = (\mathcal{Q}^{dense}, q_0, \rightarrow^{dense})$ (resp. $\mathcal{S}_{\mathcal{N}}^{discrete} = (\mathcal{Q}^{discrete}, q_0, \rightarrow^{discrete})$) being its dense-time (resp. discrete-time) semantics such that $\mathcal{Q}^{discrete} \subsetneq \mathcal{Disc}(\mathcal{Q}^{dense})$.*

Proof. Let us now exhibit an ITPN that proves this theorem. Consider the net $\mathcal{N} = (P, T, \bullet(\cdot), (\cdot)^\bullet, {}^\circ(\cdot), M_0, I)$ in figure 1.

Let us analyze its behavior. The state space of the dense-time semantics (leading to 31 different state classes) and of the discrete-time semantics can be computed.

In dense-time semantics, the following run is valid:

$$\begin{array}{ccccccc}
\{p_1, p_8\} & & \{p_1, p_8\} & & \{p_2, p_5, p_8\} & & \{p_2, p_5, p_8\} \\
\theta(guess) = 0 & \xrightarrow{0.5} & \theta(guess) = 0.5 & \xrightarrow{guess} & \begin{array}{l} \theta(c) = 0 \\ \theta(s) = 0 \\ \theta(r) = 0.5 \end{array} & \xrightarrow{3.5} & \begin{array}{l} \theta(c) = 3.5 \\ \theta(s) = 3.5 \\ \theta(r) = 4 \end{array} \xrightarrow{r} \\
\theta(r) = 0 & & \theta(r) = 0.5 & & \theta(r) = 0.5 & & \theta(r) = 4
\end{array}$$

$$\begin{array}{ccccccc}
\{p_2, p_5, p_7, p_9\} & & \{p_2, p_5, p_7, p_9\} & & \{p_2, p_6, p_7, p_9\} & & \{p_2, p_9\} \\
\theta(c) = 3.5 & \xrightarrow{0.5} & \theta(c) = 3.5 & \xrightarrow{s} & \theta(c) = 3.5 & \xrightarrow{flush} & \theta(c) = 3.5 \\
\theta(s) = 3.5 & & \theta(s) = 4 & & \theta(t) = 0.5 & & \theta(t) = 0.5 \\
\theta(t) = 0 & & \theta(t) = 0.5 & & \theta(flush) = 0 & & \theta(t) = 0.5
\end{array}$$

$$\begin{array}{ccccccc}
\{p_2, p_9\} & \xrightarrow{c} & \{p_3, p_9\} & \xrightarrow{3} & \{p_3, p_9\} & \xrightarrow{t} & \{p_3, p_{10}\} \\
\theta(c) = 4 & & \theta(u) = 0 & & \theta(u) = 3 & & \theta(u) = 3 \\
\theta(t) = 1 & & \theta(t) = 1 & & \theta(t) = 4 & & \theta(test) = 0
\end{array}$$

$$\begin{array}{ccc}
\{p_4, p_{10}\} & & \{p_4, p_{11}\} \\
\theta(test) = 1 & \xrightarrow{test} & \theta(too\_early) = 0 \\
\theta(too\_early) = 0 & & \theta(too\_late) = 0
\end{array}$$

$$\begin{array}{ccc}
& & \xrightarrow{in\_time} & \{p_{goal}\} \\
& & \theta(in\_time) = 0 &
\end{array}$$

We aim to prove that, contrary to the dense-time behavior, the analysis of the discrete-time net never leads to the firing of *in_time*. To achieve this goal, we just have to enumerate all the possible runs of the discrete-time net. This is quite easy:

- When *guess* is fired at 0, all the resulting runs end by the firing of *too_early*;
- *A contrario*, when *guess* is fired at 1, all the resulting runs end by the firing of *too_late*.

The specificity of the net we propose lies in the inhibitor arc that stops the stopwatch associated to transition *c* as long as transition *flush* (thus *s*, as *flush* fires in 0 time) does not fire. This inhibition causes the temporal shift between the upper part of the net and the bottom one (they both have a similar structure): if *guess* fires at 0, then we can say that the upper part is in advance compared to the bottom one. On the contrary, when *guess* fires at 1, the upper part is delayed. The only way to ensure that the two parts evolve in phase is that *guess* fires at 0.5 : the structure of the net then guarantees a parallel and synchronous evolution of the two branches so that *p₄* and *p₁₁* are marked simultaneously, thus allowing the firing of *in_time*.

2.3 A Sufficient Condition on ITPNs Such That the Discretization of the State Space of the Dense-Time Net and the State Space of the Discrete-Time Associated Net Coincide

For $\theta \in \mathbb{R}^+$, we denote by $frac(\theta)$ the fractional part of θ . Let $\Theta = [\theta_1 \dots \theta_n]^\top$ be a point of $(\mathbb{R}^+)^n$, we denote by $\lceil \Theta \rceil$ the point $[\lceil \theta_1 \rceil \dots \lceil \theta_n \rceil]^\top$ and $\lfloor \Theta \rfloor$ the point $\lfloor \theta_1 \rfloor \dots \lfloor \theta_n \rfloor^\top$.

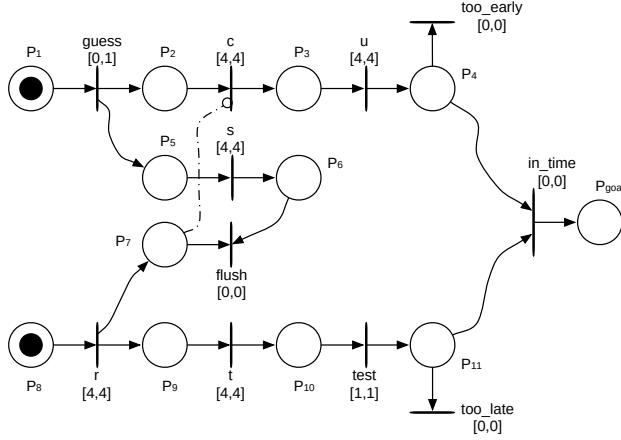


Fig. 1. ITPN showing the problems with regard to discrete-time TPNs with stopwatches

Definition 11 (t-thickness). A class $C = (M, D)$ of an ITPN $\mathcal{N}$ is said to be *t-thick* if for all transitions t_f that are firable from C , for all $\Theta = [\theta_1 \dots \theta_n]^\top \in D$ s.t. for all active transitions t_i , $\text{frac}(\theta_i) = \text{frac}(\theta_f)$ and for all inactive transitions t_j , $\text{frac}(\theta_j) = 0$, we have: either $\lceil \Theta \rceil \in D$ or $\lfloor \Theta \rfloor \in D$.

Theorem 3. Let $\mathcal{N}$ be a ITPN. Let $C = (M, D)$ be one of the state classes of its dense-time semantics $\mathcal{S}_{\mathcal{N}}^{\text{dense}}$. Let t_f be a firable transition from C . If (C, D) is *t-thick*, then $\text{next}_{t_f}^{\text{discrete}}(\text{Disc}(D)) = \text{Disc}(\text{next}_{t_f}^{\text{dense}}(D))$

Proof. The inclusion $\text{next}_{t_f}^{\text{discrete}}(\text{Disc}(D)) \subseteq \text{Disc}(\text{next}_{t_f}^{\text{dense}}(D))$ is straightforward. We shall now prove the reverse inclusion.

Let $C' = (M', D')$ be a state class of the TPN with stopwatches $\mathcal{N}$. Let $C = (M, D)$ be its parent class by the firing of transition t_f .

Let $\Theta = [\theta_1 \dots \theta_m]^\top$ be a point of D such that some transition t_f ($f \leq m$) is firable from Θ (i.e. $\forall i \leq m$, st t_i is active, $\theta_f \leq \theta_i$).

To keep the notations simple, let us assume, without loss of generality, that transitions $t_1, \dots, t_m$ are enabled by M (corresponding to variables $\theta_1 \dots \theta_m$ in D) and that the firing of t_f disables transitions $t_1 \dots t_{p-1}$ and newly enables transitions $t_{m+1} \dots t_n$. We also suppose that transitions $t_{p+1}, \dots, t_k$ with $k \leq n$ are inhibited by M and transitions $t_{k+1}, \dots, t_m$ are not.

Then,

$$\text{next}_{t_f}^{\text{dense}}(\{\Theta\}) = \left\{ [\theta'_1 \dots \theta'_{n-p+1}]^\top = \begin{cases} \forall i \in [1..k], \theta'_i = \theta_{p+i} \\ \forall i \in [k+1..m-p], \theta'_i = \theta_{p+i} - \theta_f \\ \forall i \in [m-p+1..n-p+1], \theta'_i \in I(t_i) \end{cases} \right\}$$

Let Θ' be an integer point of D' : $\Theta' \in \text{Disc}(\text{next}_{t_f}^{\text{dense}}(\{\Theta\}))$ for some $\Theta \in D$. Then, $\forall i, \theta'_i \in \mathbb{N}$, which implies $\text{frac}(\theta_f) = \text{frac}(\theta_i)$, $\forall i$ s.t. t_i is active in C

($i \in [k + 1..m - p]$), and $\text{frac}(\theta_i) = 0$ otherwise. As a consequence, since (C, D) is t -thick, either $\lceil \Theta \rceil$ or $\lfloor \Theta \rfloor$ is in D . Let us assume, as both cases are symmetric, that $\lceil \Theta \rceil \in D$.

Since t_f is firable from Θ , we have $\forall i \in [1..m]$, st t_i is active, $\theta_f \leq \theta_i$ and then $\theta_f + \delta \leq \theta_i + \delta$ for any δ , in particular for $\delta = 1 - \text{frac}(\theta_f) = 1 - \text{frac}(\theta_i)$. So t_f is firable from $\lceil \Theta \rceil$.

We have then:

$$\text{next}_{t_f}^{\text{dense}}(\{\lceil \Theta \rceil\}) = \left\{ [\theta'_1 \dots \theta'_{n-p+1}]^\top = \begin{cases} \forall i \in [1..k], \theta'_i = \lceil \theta_{p+i} \rceil \\ \forall i \in [k + 1..m - p], \theta'_i = \lceil \theta_{p+i} \rceil - \lceil \theta_f \rceil \\ \forall i \in [m - p + 1..n - p + 1], \theta'_i \in I(t_i) \end{cases} \right\}$$

For $i \in [k + 1..m - p]$, $\text{frac}(\theta_f) = \text{frac}(\theta_{p+i})$. So we have $\lceil \theta_{p+i} \rceil - \lceil \theta_f \rceil = \theta_{p+i} - \theta_f$.

For $i \in [1..k]$, $\text{frac}(\theta_i) = 0$. So we have $\lceil \theta_i \rceil = \theta_i$.

Finally, $\text{next}_{t_f}^{\text{dense}}(\{\lceil \Theta \rceil\}) = \text{next}_{t_f}^{\text{dense}}(\{\Theta\})$. As $\lceil \Theta \rceil$ is an integer point of D , we have $\text{Disc}(\text{next}_{t_f}^{\text{dense}}(\{\lceil \Theta \rceil\})) = \text{next}_{t_f}^{\text{discrete}}(\{\lceil \Theta \rceil\})$. Therefore, $\text{Disc}(\text{next}_{t_f}^{\text{dense}}(\{\Theta\})) \in \text{next}_{t_f}^{\text{discrete}}(\text{Disc}(D))$. $\square$

Theorem 4. Let $C = (M, D)$ be a class of an ITPN such that D is a DBM. C is t -thick.

Proof. All constraints in a DBM are either of the form $\theta \leq d$ or of the form $\theta - \theta' \leq d$, with d being an integer. Let $C = (M, D)$ be a class of an ITPN such that D is a DBM. Let t_f be a transition firable from C . Let $\Theta = [\theta_1 \dots \theta_n]^\top \in D$ s.t. for all active transitions t_i , $\text{frac}(\theta_i) = \text{frac}(\theta_f)$ and for all inactive transitions t_j , $\text{frac}(\theta_j) = 0$.

Due to the particular form of constraints in DBM, it is sufficient to consider the two-dimensional projections of D for our reasoning. So, let us consider any two-dimensional plane corresponding to variables θ_i and θ_j . An example of the projection of a DBM on such a plane is given in Fig 2.

1. If both θ_i and θ_j correspond to inhibited transitions, then due to the above constraints the projection of Θ must be an integer point.
2. If only θ_j corresponds to an inhibited transition, then the projection of Θ must be on one of the verticals (Fig 2(b)).
3. If both variables correspond to active transitions, then the projection of Θ must be on one of the diagonals (Fig 2(a)).

In the first case, the projection of $\lceil \Theta \rceil$ is the same as the projection of Θ so it is inside the projection of the DBM. So let us consider the other two cases.

Suppose that some constraint $\theta_i - \theta_j \leq d$ intersects the segment formed by the projections of Θ and $\lceil \Theta \rceil$. In case 3, the constraint must be parallel to the segment. Since the constraints are weak, this constraint is not excluding the projection of $\lceil \Theta \rceil$. In case 2, the intersections of all verticals with diagonal constraints are obviously integer points, so again the constraint cannot exclude $\lceil \Theta \rceil$.

The case of the constraint $\theta_i \leq d$ is similar. $\square$

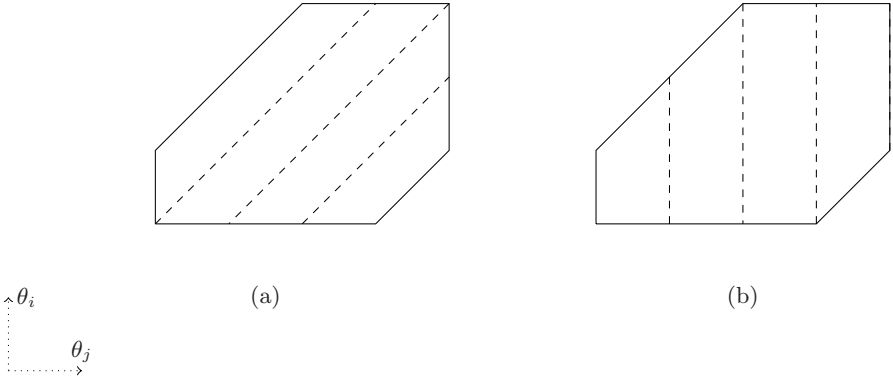


Fig. 2. Projection on a plane of a DBM: (a) θ_i and θ_j both correspond to active transitions (b) θ_j corresponds to an inhibited transition

An immediate corollary of theorems 3 and 4 is the following generalization of theorem 1:

Corollary 2. *Let $\mathcal{N}$ be a ITPN. Let $C = (M, D)$ be one of the state classes of its dense-time semantics $\mathcal{S}_{\mathcal{N}}^{\text{dense}}$ such that D is a DBM. Let t_f be a firable transition from C . Then, $\text{next}_{t_f}^{\text{discrete}}(\text{Disc}(D)) = \text{Disc}(\text{next}_{t_f}^{\text{dense}}(D))$*

In [20], we exhibited a proper TPN with stopwatches such that the firing domain of all its dense-time state classes are simple DBMs. So the class of ITPNs for which the discretization of the dense classes is exact is non-empty.

3 Symbolic Approach for the Computation of the State Space of Discrete-Time Time Petri Nets with Stopwatches

In order to build a symbolic method for state space computation in discrete-time, we need to define the notion of *symbolic state classes* for discrete-time TPNs (with stopwatches):

Definition 12. *Let $\mathcal{N} = (P, T, \bullet(\cdot), (\cdot)\bullet, \circ(\cdot), M_0, I)$ be a TPN (with or without inhibitor arcs), $n \in \mathbb{N}$ and $C^{\text{symb}} = (M, \text{Poly})$ where $M \in \mathbb{N}^P$ and Poly is a convex polyhedron of $\mathbb{R}^n$. $C = (M, \text{Poly})$ is a symbolic state class of the discrete-time semantics $\mathcal{S}_{\mathcal{N}}^{\text{discrete}}$ of the net if, for all $\nu \in \text{Disc}(\text{Poly})$, (M, ν) is a state of $\mathcal{S}_{\mathcal{N}}^{\text{discrete}}$.*

We are going to symbolically compute the state space of discrete-time nets by using these symbolic state classes. At the end of the computation process, we get a set of symbolic state classes: the discretization of this set gives the state space of the discrete-time net.

3.1 The Case of Discrete-Time TPNs

Let $\mathcal{N}$ be a TPN. We first compute the state space of $\mathcal{N}$ with its dense-time semantics $\mathcal{S}_{\mathcal{N}}^{dense}$. The discretization of each class identified during the computation leads to the state space of $\mathcal{S}_{\mathcal{N}}^{discrete}$. This results from the theorems we give in section 2.1.

3.2 The Case of Discrete-Time TPNs with Stopwatches

Let $\mathcal{N}$ be a ITPN. We aim to compute the state space of its discrete-time semantics $\mathcal{S}_{\mathcal{N}}^{discrete}$. Let us consider (M_0, D_0) the initial state class of $\mathcal{N}$ with its discrete-time semantics $\mathcal{S}_{\mathcal{N}}^{discrete}$. It is obviously a symbolic state class of $\mathcal{S}_{\mathcal{N}}^{discrete}$ that we denote C_0^{symb} . We compute the successors of this class the same way we would compute its successors for the associated dense-time model. We repeat the process as long as the on-the-fly computed state space do not need general non-DBM polyhedra to be described. As soon as a non-DBM polyhedron appears in the firing domain $Poly$ of a state class $C^{poly} = (M^{poly}, Poly)$, then we decompose it into a union of DBMs $DBM\_split(Poly) = \bigcup D_i^{split}$. In fact, in [20], we identified a necessary and sufficient condition (this condition is quite long; so, in the following, we denote it simply by condition 3.2 of [20]) that establishes the cases when a non-DBM polyhedron appears in the state space computation for a dense-time model. So we use this condition to know when we have to split the polyhedron into a union of DBMs.

The splitting procedure $DBM\_split(.)$ of a polyhedron into a union of DBMs (with preservation of the property $Disc(Poly) = \bigcup Disc(D_i^{split})$) is not unique.

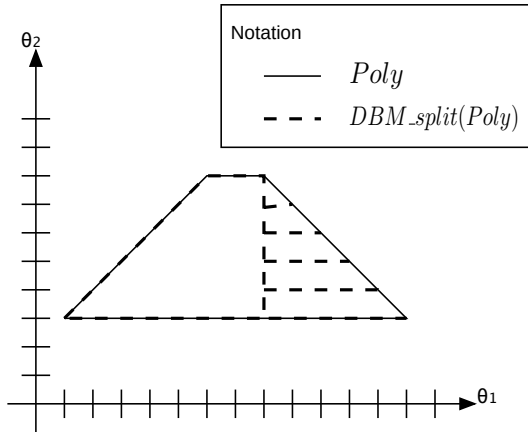


Fig. 3. Illustration of the effects of a splitting procedure. $Poly$ represents the temporal domain associated to a symbolic class of a discrete-time ITPN. $DBM\_split(Poly)$ corresponds to a potential decomposition of this polyhedron into a union of DBMs such that $Disc(Poly) = Disc(DBM\_split(Poly))$.

A rather obvious (but really not efficient) algorithm consists in decomposing the polyhedron $Poly$ into the union of all its integer points $Disc(Poly)$. A more subtle approach consists in decomposing the polyhedron according to the variables that are part of non-DBM constraints of the polyhedron. In figure 3, we illustrate the issues related to $DBM\_split(\cdot)$: the problem is to split the polyhedron $Poly$ into a union of DBMs with preservation of discrete-time points. The solution to this solution is not unique; the one we propose appears with dashed lines. Many other solutions may be considered but this study is out of the scope of this paper.

Algorithm 1: Symbolic algorithm for the computation of the state space of discrete-time TPNs with stopwatches

```

Passed =  $\emptyset$ 
Waiting =  $\{(M_0, D_0)\}$ 
While (Waiting  $\neq \emptyset$ ) do
   $(M, D) = pop(Waiting)$ 
   $Passed = Passed \cup (M, D)$ 
  For  $t_f$  firable from  $(M, D)$  do
     $M' = M - \bullet t_f + t_f^\bullet$ 
    If (condition 3.2 of [20] is not satisfied) then
       $D' = next^{dense}(D, t_f)$  [5]
      If  $((M', D') \notin Passed)$  then
         $Waiting = Waiting \cup \{(M', D')\}$ 
      end If
    else
       $\bigcup (D'_i)_{i \in [1, \dots, n]} = DBM\_split(next^{dense}(D, t_f)$  [5])
      For  $i \in [1, \dots, n]$  do
        If  $((M', D'_i) \notin Passed)$  then
           $Waiting = Waiting \cup \{(M', D'_i)\}$ 
        end If
      end For
    end If
  end For
done

```

This method is summarized by the formal algorithm 1.

Theorem 5. *For discrete-time ITPNs, the algorithm 1 is correct w.r.t. marking and state reachability and language. The termination is ensured for discrete-time bounded ITPNs.*

To prove this algorithm, we first have to introduce a corollary of theorem 3.

In [20], we studied the conditions such that general polyhedra (that cannot be written under the form of simple DBMs) appear in the state class graph of dense-time TPNs with stopwatches. We then deduce the following corollary:

Corollary 3. *Let $\mathcal{N}$ be a ITPN. We aim to determine the state space of the discrete-time semantics $\mathcal{S}_{\mathcal{N}}^{discrete}$ of the net by linking it to the on-the-fly computation of the state space of $\mathcal{S}_{\mathcal{N}}^{dense}$. As long as no class of $\mathcal{S}_{\mathcal{N}}^{dense}$ satisfies the necessary and sufficient condition (condition 3.2) exhibited in [20], we have the following properties:*

- *The discretization of the resulting state space of $\mathcal{S}_{\mathcal{N}}^{dense}$ gives states that all belong to the state space of $\mathcal{S}_{\mathcal{N}}^{discrete}$.*
- *Every untimed run of $\mathcal{S}_{\mathcal{N}}^{dense}$ that is identified is included in the set of all untimed runs of $\mathcal{S}_{\mathcal{N}}^{discrete}$.*

Then we deduce the correctness of the algorithm:

Proof. The correctness of algorithm 1 follows from the previous theorems of our paper, especially from theorem 3 and corollary 3. The convergence of the algorithm is a consequence of the decidability of the reachability problem for discrete-time bounded ITPNs.

Such an abstraction has many practical implications: it enables us to use algorithms and tools based on DBMs developed for TPNs to check properties on ITPNs with discrete time semantics. We use the tool ROMÉO [21] that has been developed for the analysis of TPN (state space computation and "on the fly" model-checking of reachability properties and TCTL properties). For instance, it is possible to check real-time properties expressed in TCTL on bounded discrete-time ITPNs by a very simple adaptation of this tool.

On the one hand, in [22], the authors consider TCTL model-checking on TPNs. In fact, their algorithms apply on all Petri nets extended with time such that the firing domains of all state classes are DBMs. On the other hand, algorithm 1 states how to compute the state space of discrete-time ITPNs by using only DBMs. The combination of the two procedures leads to an elegant way to model-check TCTL formulae on discrete-time ITPNs. The implementation in ROMÉO leads to quite nice results. To illustrate the merits of our work, we provide a benchmark that compares the efficiency, in terms of computation speed, of the state space computation using the enumerative technique we introduced in [8] with the symbolic algorithm we propose in this paper.

All examples depict bounded nets. These nets correspond to academical cases relevant to the matter of our studies. When the intervals associated to transitions are small (example 2), the enumerative method is more efficient than the symbolic algorithm. This is due to the power of BDDs based techniques available on the enumerative approach. Nevertheless, when the net contains one (or more) transitions with a wide range between earliest and latest firing times (all examples, except the second one, contain a transition whose firing interval is $[0, 1000]$), the enumerative method suffers from combinatorial explosion, while our symbolic algorithm leads to good results. While the magnitude of timing delays affects the enumerative approach, the use of numerous inhibitor arcs may decrease the efficiency of the symbolic algorithm and make it preferable to perform enumerative algorithms. If a net contains various transitions that

Table 1. Comparison between symbolic algorithm (with ROMÉO) and enumerative techniques (with Markg) to compute the state space of discrete-time nets (PENTIUM ; 2 GHz; 2GB RAM)

Net	Symbolic algorithm - ROMÉO		Enumerative algorithm - Markg	
	Time	Memory	Time	Memory
Ex 1	0.12 s	1 320 KB	1.03 s	96 032 KB
Ex 2	34.42 s	1 320 KB	2.95 s	111 700 KB
Ex 3	45.12 s	20 012 KB	NA	NA
Ex 4	0.52 s	2 940 KB	1.07 s	95 796 KB
Ex 5	0.15 s	1 320 KB	1 148.18 s	139 800 KB

may be *successively* inhibited while other transitions are enabled and active, it is necessary to often split polyhedra into unions of DBMs. This results in a loss of efficiency of the symbolic method. Such cases become significant when the number of simultaneously enabled transitions is greater than 10. Further work encompasses tests on true case studies taken out from literature about real-time systems.

4 Conclusion

In this paper, we have considered an extension of T-time Petri nets with stop-watches (SwPNs). In this model, stopwatches are associated with transitions: a stopwatch can be reset, stopped and started by using inhibitor arcs. This allows the memorisation of the progress status of an action that is stopped and re-sumed. Our methods and results have been illustrated on Time Petri Nets with Inhibitor arcs (ITPNs). They are however general enough to be applied on the whole class of SwPNs.

We aimed to extend to discrete-time semantics the symbolic methods usually applied for the computation of the state space of a dense-time TPN (with stop-watches) and then perform TCTL model-checking. The approach we introduce consists in linking the state space of a discrete-time net to the discretization of the state space of the associated dense-time model. This method is correct for a subclass of SwPNs (including TPNs), but not for the general class of SwPNs. We thus propose a more general method for computing symbolically the state space of discrete-time nets. It is based on the decomposition of the general polyhedra that encompass the temporal information of the net into a union of simpler polyhedra. The subtlety of the symbolic algorithm depends on the function that splits the polyhedra into a union of DBMs: shall the polyhedron be split in a minimum number of DBMs? Are some splitting procedure more efficient than others for a long-term computation? What is the cost of such algorithms? These questions are the basis of our current investigations.

Further work consist in investigating the efficiency of several algorithms that split a general polyhedra in unions of DBMs.

References

1. Merlin, P.: A study of the recoverability of computing systems. PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA (1974)
2. Ramchandani, C.: Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA Project MAC Report MAC-TR-120 (1974)
3. Roux, O., Déplanche, A.M.: A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)* 36, 973–987 (2002)
4. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Time state space analysis of real-time preemptive systems. *IEEE transactions on software engineering* 30, 97–111 (2004)
5. Roux, O.H., Lime, D.: Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 371–390. Springer, Heidelberg (2004)
6. Berthomieu, B., Lime, D., Roux, O.H., Vernadat, F.: Reachability problems and abstract state spaces for time petri nets with stopwatches. *Journal of Discrete Event Dynamic Systems (DEDS)* (to appear, 2007)
7. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Automata, Languages and Programming, pp. 545–558 (1992)
8. Magnin, M., Molinaro, P., Roux, O.H.: Decidability, expressivity and state-space computation of stopwatch petri nets with discrete-time semantics. In: 8th International Workshop on Discrete Event Systems (WODES 2006), Ann Arbor, USA (2006)
9. Popova, L.: On time petri nets. *Journal Inform. Process. Cybern., EIK (formerly: Elektron. Inform. verarb. Kybern)* 27, 227–244 (1991)
10. Popova-Zeugmann, L.: Essential states in time petri nets (1998)
11. Popova-Zeugmann, L., Schlatter, D.: Analyzing paths in time petri nets. *Fundamenta Informaticae* 37, 311–327 (1999)
12. Molinaro, P., Delfieu, D., Roux, O.H.: Improving the calculus of the marking graph of Petri net with bdd like structure. In: 2002 IEEE international conference on systems, man and cybernetics (SMC 2002), Hammamet, Tunisia (2002)
13. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering* 17, 259–273 (1991)
14. Gardey, G., Roux, O.H., Roux, O.F.: State space computation and analysis of time Petri nets. Theory and Practice of Logic Programming (TPLP) (to appear, 2006); Special Issue on Specification Analysis and Verification of Reactive Systems
15. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time petri nets. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 442–457. Springer, Heidelberg (2003)
16. Boucheneb, H., Gardey, G., Roux, O.H.: TCTL model checking of time Petri nets. Technical Report IRCCyN number RI2006-14 (2006)
17. Lime, D., Roux, O.: Expressiveness and analysis of scheduling extended time Petri nets. In: 5th IFAC International Conference on Fieldbus Systems and their Applications (FET 2003), Aveiro, Portugal. Elsevier Science, Amsterdam (2003)
18. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time Petri nets. *IFIP Congress Series* 9, 41–46 (1983)
19. Dill, D.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)

20. Magnin, M., Lime, D., Roux, O.: An efficient method for computing exact state space of Petri nets with stopwatches. In: Third International Workshop on Software Model-Checking (SoftMC 2005), Edinburgh, Scotland, UK. Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam (2005)
21. Gardey, G., Lime, D., Magnin, M., Roux, O.H.: Roméo: A tool for analyzing time Petri nets. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 418–423. Springer, Heidelberg (2005)
22. Hadjidj, R., Boucheneb, H.: On-the-fly tctl model checking for time petri nets using state class graphs. In: ACSD, pp. 111–122. IEEE Computer Society, Los Alamitos (2006)

A Practical Approach to Verification of Mobile Systems Using Net Unfoldings

Roland Meyer¹, Victor Khomenko², and Tim Strazny¹

¹ Department of Computing Science, University of Oldenburg
D-26129 Oldenburg, Germany

{Roland.Meyer, Tim.Strazny}@informatik.uni-oldenburg.de

² School of Computing Science, University of Newcastle
Newcastle upon Tyne, NE1 7RU, U.K.

Victor.Khomenko@ncl.ac.uk

Abstract. In this paper we propose a technique for verification of mobile systems. We translate *finite control processes*, which are a well-known subset of π -Calculus, into Petri nets, which are subsequently used for model checking. This translation always yields bounded Petri nets with a small bound, and we develop a technique for computing a non-trivial bound by static analysis. Moreover, we introduce the notion of *safe processes*, which are a subset of finite control processes, for which our translation yields safe Petri nets, and show that every finite control process can be translated into a safe one of at most quadratic size. This gives a possibility to translate every finite control process into a safe Petri net, for which efficient unfolding-based verification is possible. Our experiments show that this approach has a significant advantage over other existing tools for verification of mobile systems in terms of memory consumption and runtime.

Keywords: finite control processes, safe processes, π -Calculus, mobile systems, model checking, Petri net unfoldings.

1 Introduction

Mobile systems permeate our lives and are becoming ever more important. Ad-hoc networks, where devices like mobile phones, PDAs and laptops form dynamic connections are common nowadays, and the vision of pervasive (ubiquitous) computing, where several devices are simultaneously engaged in interaction with the user and each other, forming dynamic links, is quickly becoming a reality. This leads to the increasing dependency of people on the correct functionality of mobile systems, and to the increasing cost incurred by design errors in such systems. However, even the conventional concurrent systems are notoriously difficult to design correctly because of the complexity of their behaviour, and mobile systems add another layer of complexity due to their dynamical nature. Hence formal methods, especially computer-aided verification tools implementing model checking (see, e.g., [CGP99]), have to be employed in the design process to ensure correct behaviour.

The π -Calculus is a well-established formalism for modelling mobile systems [Mil99, SW01]. It has an impressive modelling power, but, unfortunately, is difficult to verify. The full π -Calculus is Turing complete, and hence, in general, intractable for automatic techniques. A common approach is to sacrifice a part of the modelling power of π -Calculus in exchange for the possibility of fully automatic verification. Expressive fragments of π -Calculus have been proposed in the literature. In particular *finite control processes (FCPs)* [Dam96] combine an acceptably high modelling power with the possibility of automatic verification.

In this paper, we propose an efficient model checking technique for FCPs. We translate general FCPs into their syntactic subclass, called *safe processes*. In turn, safe processes admit an efficient translation into safe Petri nets — a well-investigated model for concurrent systems, for which efficient model checking techniques have been developed.

This approach has a number of advantages, in particular it does not depend on a concrete model checking technique, and can adapt any model checker for safe Petri nets. Moreover, Petri nets are a *true concurrency* formalism, and so one can efficiently utilise partial-order techniques. This alleviates the main drawback of model checking — the *state explosion* problem [Val98]; that is, even a small system specification can (and often does) yield a huge state space.

Among partial-order techniques, a prominent one is McMillan's (finite prefixes of) Petri Net unfoldings (see, e.g., [ERV02, Kho03, McM92]). They rely on the partial-order view of concurrent computation, and represent system states implicitly, using an acyclic net, called a *prefix*. Many important properties of Petri nets can be reformulated as properties of the prefix, and then efficiently checked, e.g., by translating them to SAT. Our experiments show that this approach has a significant advantage over other existing tools for verification of mobile systems in terms of memory consumption and runtime. The proofs of the results and other technical details can be found in the technical report [MKS08].

2 Basic Notions

In this section, we recall the basic notions concerning π -Calculus and Petri nets.

The π -Calculus. We use a π -Calculus with parameterised recursion as proposed in [SW01]. Let the set $\mathcal{N} \stackrel{\text{df}}{=} \{a, b, x, y, \dots\}$ of *names* contain the channels (which are also the possible messages) that occur in communications. During a process execution the *prefixes* π are successively consumed (removed) from the process to communicate with other processes or to perform silent actions:

$$\pi ::= \bar{a}(b) \mid a(x) \mid \tau.$$

The *output action* $\bar{a}(b)$ sends the name b along channel a . The *input action* $a(x)$ receives a name that replaces x on a . The τ prefix stands for a *silent action*.

To denote recursive processes, we use *process identifiers* from the set $PIDS \stackrel{\text{df}}{=} \{H, K, L, \dots\}$. A process identifier is defined by an equation $K(\tilde{x}) := P$, where $\tilde{x}$ is a short-hand notation for $x_1, \dots, x_k$. When the identifier is *called*, $K[\tilde{a}]$, it is replaced by the process obtained from P by replacing the names $\tilde{x}$ by $\tilde{a}$. More

precisely, a *substitution* $\sigma = \{\tilde{a}/\tilde{x}\}$ is a function that maps the names in $\tilde{x}$ to $\tilde{a}$, and is the identity for all the names not in $\tilde{x}$. The *application of substitution*, $P\sigma$, is defined in the standard way [SW01]. A π -Calculus process is either a call to an identifier, $K[\tilde{a}]$, a *choice process* deciding between prefixes, $\sum_{i \in I} \pi_i.P_i$, a *parallel composition* of processes, $P_1 \mid P_2$, or the *restriction* of a name in a process, $\nu a.P$:

$$P ::= K[\tilde{a}] \mid \sum_{i \in I} \pi_i.P_i \mid P_1 \mid P_2 \mid \nu a.P.$$

The *set of all processes* is denoted by $\mathcal{P}$. We abbreviate empty sums (i.e., those with $I = \emptyset$) by $\mathbf{0}$ and use M or N to denote arbitrary sums. We also use the notation $\Pi_{i=1}^n P_i$ for iterated parallel composition. Processes that do not contain the parallel composition operator are called *sequential*. We denote sequential processes by P_S , Q_S and the identifiers they use by K_S . An identifier K_S is defined by $K_S(\tilde{x}) := P_S$ where P_S is a sequential process. W.l.o.g., we assume that every process either is $\mathbf{0}$ or does not contain $\mathbf{0}$. To see that this is no restriction consider the process $\bar{a}\langle b \rangle.\mathbf{0}$. We transform it to $\bar{a}\langle b \rangle.K[-]$ with $K(-) := \mathbf{0}$.

The input action $a(b)$ and the restriction $\nu c.P$ bind the names b and c , respectively. The *set of bound names* in a process P is $bn(P)$. A name which is not bound is *free*, and the *set of free names* in P is $fn(P)$. We permit α -conversion of bound names. Therefore, w.l.o.g., we assume that a name is bound at most once in a process and $bn(P) \cap fn(P) = \emptyset$. Moreover, if a substitution $\sigma = \{\tilde{a}/\tilde{x}\}$ is applied to a process P , we assume $bn(P) \cap (\tilde{a} \cup \tilde{x}) = \emptyset$.

We use the *structural congruence* relation in the definition of the behaviour of a process term. It is the smallest congruence where α -conversion of bound names is allowed, $+$ and $\mid$ are commutative and associative with $\mathbf{0}$ as the neutral element, and the following laws for restriction hold:

$$\nu x.\mathbf{0} \equiv \mathbf{0} \quad \nu x.\nu y.P \equiv \nu y.\nu x.P \quad \nu x.(P \mid Q) \equiv P \mid (\nu x.Q), \text{ if } x \notin fn(P).$$

The last rule is called *scope extrusion*. The behaviour of π -Calculus processes is then determined by the *reaction relation* $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ defined by:

$$\begin{aligned} (\text{Par}) \quad & \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} & (\text{Tau}) \quad & \tau.P + M \rightarrow P & (\text{Res}) \quad & \frac{P \rightarrow P'}{\nu a.P \rightarrow \nu a.P'} \\ (\text{React}) \quad & (x(y).P + M) \mid (\bar{x}\langle z \rangle.Q + N) \rightarrow P\{z/y\} \mid Q \\ (\text{Const}) \quad & K[\tilde{a}] \rightarrow P\{\tilde{a}/\tilde{x}\}, \text{ if } K(\tilde{x}) := P \\ (\text{Struct}) \quad & \frac{P \rightarrow P'}{Q \rightarrow Q'}, \text{ if } P \equiv Q \text{ and } P' \equiv Q'. \end{aligned}$$

By *Reach*(P) we denote the *set of all processes reachable from P* by the reaction relation. We use a client-server system to illustrate the behaviour of a π -Calculus process. It will serve us as a running example throughout the paper.

Example 1. Consider the process $C[url] \mid C[url] \mid S[url]$ modelling two clients and a sequential server, with the corresponding process identifiers defined as

$$\begin{aligned} C(url) &:= \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C[url] \\ S(url) &:= url(y).\nu ses.\bar{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S[url]. \end{aligned}$$

The server is located at some URL, $S[url]$. To contact it, a client sends its ip address on the channel url , $\overline{url}\langle ip \rangle$. This ip address is different for every client, therefore it is restricted. The server receives the IP address of the client and stores it in the variable y , $url(y)$. To establish a private connection with the client, the server creates a temporary session, νses , which it passes to the client, $\overline{y}\langle ses \rangle$. Note that by rule (React), y is replaced by ip during the system execution. Thus, the client receives this session, $ip(s)$. Client and server then continue to interact, which is not modelled explicitly. At some point, the server decides that the session should be ended. It sends the session object itself to the client, $\overline{ses}\langle ses \rangle$, and becomes a server again, $S[url]$. The client receives the message, $s(x)$, and calls its recursive definition to be able to contact the server once more, $C[url]$. The model can contain several clients (two in our case), but the server is engaged with one client at a time. $\diamond$

Our theory employs a standard form of process terms, the so-called *restricted form* [Mey07]. It minimises the scopes of all restricted names νa not under a prefix π . Then processes congruent with $\mathbf{0}$ are removed. For example, the restricted form of $P = \nu a.\nu d.(\overline{a}\langle b \rangle.Q \mid \overline{b}\langle c \rangle.R)$ is $\nu a.\overline{a}\langle b \rangle.Q \mid \overline{b}\langle c \rangle.R$, but the restricted form of $\overline{a}\langle b \rangle.P$ is $\overline{a}\langle b \rangle.P$ itself. A *fragment* is a process of the form

$$F ::= K[\tilde{a}] \mid \sum_{i \in I \neq \emptyset} \pi_i.P_i \mid \nu a.(F_1 \mid \dots \mid F_n),$$

where $P_i \in \mathcal{P}$ and $a \in fn(F_i)$ for all i . We denote fragments by F or G . A process P_ν is in the *restricted form*, if it is a parallel composition of fragments, $P_\nu = \Pi_{i \in I} G_i$. The *set of fragments in P_ν* is denoted by $Frag(P_\nu) \stackrel{\text{df}}{=} \{G_i \mid i \in I\}$. The *set of all processes in restricted form* is denoted by $\mathcal{P}_\nu$.

For every process P , the function $(-)_\nu$ computes a structurally congruent process $(P)_\nu$ in the restricted form [Mey07]. For a choice composition and a call to a process identifier $(-)_\nu$ is defined to be the identity, and $(P \mid Q)_\nu \stackrel{\text{df}}{=} (P)_\nu \mid (Q)_\nu$. In the case of restriction, $\nu a.P$, we first compute the restricted form of P , which is a parallel composition of fragments, $(P)_\nu = \Pi_{i \in I} F_i$. We then restrict the scope of a to the fragments F_i where a is a free name (i.e., $i \in I_a \subseteq I$): $(\nu a.P)_\nu \stackrel{\text{df}}{=} \nu a.(\Pi_{i \in I_a} F_i \mid \Pi_{i \in I \setminus I_a} F_i)$.

Lemma 1. *For every process $P \in \mathcal{P}$ it holds $(P)_\nu \in \mathcal{P}_\nu$ and $P \equiv (P)_\nu$. For $P_\nu \in \mathcal{P}_\nu$ we have $(P_\nu)_\nu = P_\nu$.*

If we restrict structural congruence to processes in restricted form, we get the *restricted equivalence* relation $\hat{\equiv}$. It is the smallest equivalence on processes in restricted form that permits (1) associativity and commutativity of parallel composition and (2) replacing fragments by structurally congruent ones, i.e., $F \mid P_\nu \hat{\equiv} G \mid P_\nu$ if $F \equiv G$. It characterises structural congruence [Mey07]:

Lemma 2. $P \equiv Q$ iff $(P)_\nu \hat{\equiv} (Q)_\nu$.

Petri Nets. A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, W)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N} \stackrel{\text{df}}{=} \{0, 1, 2, \dots\}$ is a *weight function*. A *marking* of N is a multiset M of places, i.e., $M : P \rightarrow \mathbb{N}$. The standard rules about drawing nets are adopted in this

paper, viz. places are represented as circles, transitions as boxes, the weight function by arcs with numbers (the absence of an arc means that the corresponding weight is 0, and an arc with no number means that the corresponding weight is 1), and the marking is shown by placing tokens within circles. As usual, $\bullet z \stackrel{\text{df}}{=} \{y \mid W(y, z) > 0\}$ and $z \bullet \stackrel{\text{df}}{=} \{y \mid W(z, y) > 0\}$ denote the *pre-* and *postset* of $z \in P \cup T$, and $\bullet Z \stackrel{\text{df}}{=} \bigcup_{z \in Z} \bullet z$ and $Z \bullet \stackrel{\text{df}}{=} \bigcup_{z \in Z} z \bullet$, for all $Z \subseteq P \cup T$. In this paper, the presets of transitions are restricted to be non-empty, $\bullet t \neq \emptyset$ for every $t \in T$. A *net system* is a pair $\mathcal{T} \stackrel{\text{df}}{=} (N, M_0)$ comprising a finite net N and an *initial* marking M_0 .

A transition $t \in T$ is *enabled* at a marking M , denoted $M[t]$, if $M(p) \geq W(p, t)$ for every $p \in \bullet t$. Such a transition can be *fired*, leading to the marking M' with $M'(p) \stackrel{\text{df}}{=} M(p) - W(p, t) + W(t, p)$, for every $p \in P$. We denote this by $M[t]M'$ or $M[\]M'$ if the identity of the transition is irrelevant. The set of *reachable* markings of $\mathcal{T}$ is the smallest (w.r.t. $\subseteq$) set $[M_0]$ containing M_0 and such that if $M \in [M_0]$ and $M[\]M'$ then $M' \in [M_0]$.

A net system $\mathcal{T}$ is *k-bounded* if, for every reachable marking M and every place $p \in P$, $M(p) \leq k$, and *safe* if it is 1-bounded. Moreover, $\mathcal{T}$ is *bounded* if it is *k-bounded* for some $k \in \mathbb{N}$. One can show that the set $[M_0]$ is finite iff $\mathcal{T}$ is bounded. W.l.o.g., we assume that for net systems known to be safe the range of the weight function is $\{0, 1\}$.

3 A Petri Net Translation of the π -Calculus

We recall the translation of π -Calculus processes into Petri nets defined in [Mey07]. The translation is based on the observation that processes are connected by restricted names they share. Consider the fragment $\nu a.(K[a] \mid L[a])$. As the scope of a cannot be shrunk using the scope extrusion rule, the restricted name a ‘connects’ the processes $K[a]$ and $L[a]$. The idea of the translation is to have a separate place in the Petri net for each reachable ‘bunch’ of processes connected by restricted names, i.e., the notion of fragments plays a crucial role in the proposed translation. The algorithm takes a π -Calculus process P and computes a Petri net $\mathcal{PN}[P]$ as follows:

- The places in the Petri net are all the fragments of every reachable process (more precisely, the congruence classes of fragments w.r.t. $\equiv$).
- The transitions consist of three disjoint subsets:
 - Transitions $t = ([F], [Q])$ model reactions inside a fragment F , where Q is such that $F \rightarrow Q$ and $[F]$ is a place (i.e., F is a reachable fragment). These reactions are communications of processes within F , τ actions, or calls to process identifiers, $K[\tilde{a}]$. There is an arc weighted one from place $[F]$ to t .
 - Transitions $t = ([F \mid F], [Q])$ model reactions between two structurally congruent reachable fragments along public channels, i.e., $F \mid F \rightarrow Q$ and $[F]$ is a place. There is an arc weighted two from $[F]$ to t . If this transition is fired, F contains a sequential process sending on a public

channel and another one receiving on that channel, and there are two copies (up to $\equiv$) of F in the current process.

- Transitions $t = ([F_1 \mid F_2], [Q])$ model reactions between reachable fragments $F_1 \not\equiv F_2$ along public channels: $F_1 \mid F_2 \rightarrow Q$ and $[F_1]$ and $[F_2]$ are places. There are two arcs each weighted one from $[F_1]$ and $[F_2]$ to t .

The postsets of each kind of transitions are the reachable fragments in the restricted form of Q . If the fragment G occurs (up to $\equiv$) $k \in \mathbb{N}$ times in $(Q)_\nu$, then there is an arc weighted k from $([F], [Q])$ to $[G]$. For example, from the transition $([\tau.\Pi_{i=1}^3 K[a]], [\Pi_{i=1}^3 K[a]])$ there is an arc weighted three to the place $[K[a]]$.

- The initial marking of place $[F]$ in $\mathcal{PN}[[P]]$ equals to the number of fragments in the restricted form of P that are congruent with F .

Note that if it is known in advance that the resulting Petri net will be safe, then no transition incident to an arc of weight more than one can fire, and so they can be dropped by the translation (in particular, the second kind of transitions will never appear). This fact can be used to optimise the translation of *safe processes* defined in Section 5.

It turns out that a π -Calculus process and the corresponding Petri net obtained by this translation have isomorphic transition systems [Mey07]. Hence, one can verify properties specified for a process P using $\mathcal{PN}[[P]]$. Returning to our running example, this translation yields the Petri net in Figure 1(a) for the process in Example 1.

Our translation is particularly suitable for verification because it represents an expressive class of processes (viz. FCPs) with potentially unbounded creation of restricted names as bounded Petri nets.

4 Boundedness of FCP Nets

For general π -Calculus processes, the translation presented in the previous section may result in infinite Petri nets, and even when the result is finite, it can be unbounded, which is bad for model checking. (Model checking of even simplest properties of unbounded Petri nets is EXPSPACE-hard.) To make verification feasible in practice, we need bounded nets, preferably even safe ones (the unfolding-based verification is especially efficient for safe nets), and so we have to choose an expressive subclass of π -Calculus which admits efficient verification.

In this section, we investigate the translation of the well-known *finite control processes* (FCPs), a syntactic subclass of the π -Calculus [Dam96]. FCPs are parallel compositions of a finite number of sequential processes P_{S_i} , $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$, and so new threads are never created and the degree of concurrency is bounded by n . The main result in this section states that the Petri net $\mathcal{PN}[[P_{\mathcal{FC}}]]$ is bounded, and a non-trivial bound can be derived syntactically from the structure of $P_{\mathcal{FC}}$. The intuitive idea is that k tokens on a place $[F]$ require at least k processes P_{S_i} in $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ that share some process identifiers.

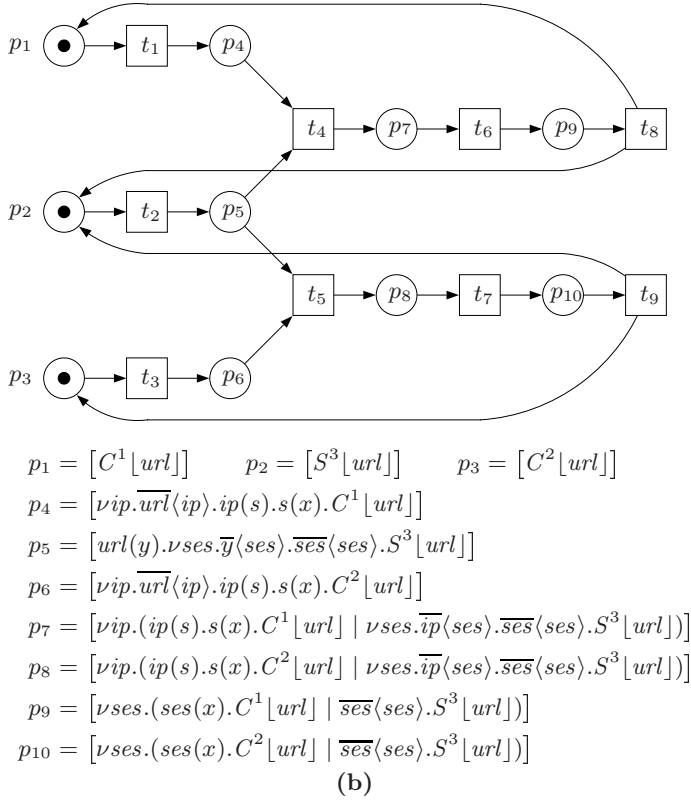
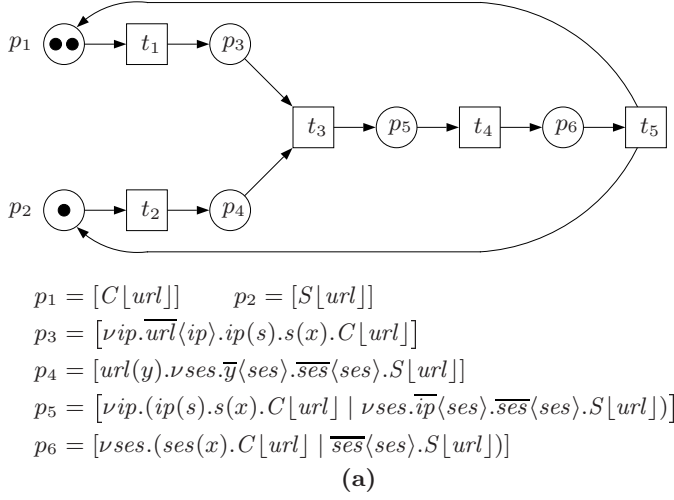


Fig. 1. The Petri nets corresponding to the FCP in Example 1 (a) and to the corresponding safe process in Example 3 (b)

To make the notion of sharing process identifiers precise we define *orbits*. The orbit of a sequential process P_{S_i} consists of the identifiers P_{S_i} calls (directly or indirectly). With this idea, we rephrase our result: if there are at most k orbits in $P_{\mathcal{FC}}$ whose intersection is non-empty then the net $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ is k -bounded.

Generally, the result states that *the bound of $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ is small*. If $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ then $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ is trivially n -bounded, as the total number of orbits is n . Often, our method yields bounds which are better than n . This should be viewed in the light of the fact that for general bounded Petri nets the bound is double-exponential in the size of the net [Esp98]. This limits the state space in our translation and makes such nets relatively easy to model check.

The intuitive idea of the *orbit function* is to collect all process identifiers syntactically reachable from a given process. We employ the function $ident : \mathcal{P} \rightarrow \mathbb{P}(PIDS)$ which gives the set of process identifiers $ident(P)$ that are in the process $P \in \mathcal{P}$:

$$\begin{aligned} ident(K[\tilde{a}]) &\stackrel{\text{df}}{=} \{K\} & ident(\sum_{i \in I} \pi_i.P_i) &\stackrel{\text{df}}{=} \bigcup_{i \in I} ident(P_i) \\ ident(\nu a.P) &\stackrel{\text{df}}{=} ident(P) & ident(P \mid Q) &\stackrel{\text{df}}{=} ident(P) \cup ident(Q). \end{aligned}$$

The *orbit* of a process P , $orb(P)$, is the smallest (w.r.t. $\subseteq$) set such that $ident(P) \subseteq orb(P)$ and if a process identifier K with a defining equation $K(\tilde{x}) := Q$ is in $orb(P)$ then $ident(Q) \subseteq orb(P)$. The *maximal number of intersecting orbits of a process $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$* is

$$\#_{\cap}(P_{\mathcal{FC}}) \stackrel{\text{df}}{=} \max \{ |I| \mid I \subseteq \{1, \dots, n\} \text{ and } \bigcap_{i \in I} orb(P_{S_i}) \neq \emptyset \}.$$

The main result of this section can now be stated as follows.

Theorem 1. $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ is $\#_{\cap}(P_{\mathcal{FC}})$ -bounded.

Example 2. Consider $P_{\mathcal{FC}} = C[url] \mid C[url] \mid S[url]$ in Example 1. We have $orb(S[url]) = \{S\}$ and $orb(C[url]) = \{C\}$ for both clients. Thus, $\#_{\cap}(P_{\mathcal{FC}}) = 2$, and so the corresponding Petri net $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ in Figure 1(a) is 2-bounded. This is an improvement on the trivial bound of 3 (i.e., the number of concurrent processes in the system). $\diamond$

We spend the rest of the section sketching the proof of this result. The Petri net $\mathcal{PN}[\llbracket P_{\mathcal{FC}} \rrbracket]$ is k -bounded iff in every reachable process $Q \in Reach(P_{\mathcal{FC}})$ there are at most k fragments that are structurally congruent. Thus, we need to show that the number of structurally congruent fragments is bounded by $\#_{\cap}(P_{\mathcal{FC}})$ in every reachable process Q . To do so, we assume there are k fragments $F_1 \equiv \dots \equiv F_k$ in Q and conclude that there are at least k intersecting orbits in $P_{\mathcal{FC}}$, i.e., $\#_{\cap}(P_{\mathcal{FC}}) \geq k$. We argue as follows. From structural congruence we know that the identifiers in all F_i are equal. We now show that the identifiers of the F_i are already contained in the orbits of different P_{S_i} in $P_{\mathcal{FC}}$. Thus, the intersection $orb(P_{S_1}) \cap \dots \cap orb(P_{S_k})$ is not empty. This means that we have found k intersecting orbits, i.e., $\#_{\cap}(P_{\mathcal{FC}}) \geq k$.

To show $ident(F_i) \subseteq orb(P_{S_i})$ we need to relate the processes in every reachable fragment with the initial process $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$. To achieve

this, we prove that every reachable process is a parallel composition of subprocesses of P_{S_i} . These subprocesses are in the set of *derivatives* of P_{S_i} , which are defined by removing prefixes from P_{S_i} as if those prefixes were consumed.

Definition 1. The function $der : \mathcal{P} \rightarrow \mathbb{P}(\mathcal{P})$ assigns to every process P the set $der(P)$ as follows:

$$\begin{aligned} der(\mathbf{0}) &\stackrel{\text{df}}{=} \emptyset & der(K[\tilde{a}]) &\stackrel{\text{df}}{=} \{K[\tilde{a}]\} \\ der(\sum_{i \in I \neq \emptyset} \pi_i.P_i) &\stackrel{\text{df}}{=} \{\sum_{i \in I \neq \emptyset} \pi_i.P_i\} \cup \bigcup_{i \in I} der(P_i) \\ der(\nu a.P) &\stackrel{\text{df}}{=} der(P) & der(P \mid Q) &\stackrel{\text{df}}{=} der(P) \cup der(Q). \end{aligned}$$

Consider an FCP $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$. We assign to every P_{S_i} the set of derivatives, $derivatives(P_{S_i})$. It is the smallest (w.r.t. $\subseteq$) set such that $der(P_{S_i}) \subseteq derivatives(P_{S_i})$ and if $K_S[\tilde{a}] \in derivatives(P_{S_i})$ then $der(P_S) \subseteq derivatives(P_{S_i})$, where $K_S(\tilde{x}) := P_S$. $\diamond$

Using structural congruence, every process reachable from $P_{\mathcal{FC}}$ can be rewritten as a parallel composition of derivatives of the processes in $P_{\mathcal{FC}}$. This technical lemma relates every reachable process with the processes in $P_{\mathcal{FC}}$.

Lemma 3. Let $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$. Then every $Q \in Reach(P_{\mathcal{FC}})$ is structurally congruent with $\nu \tilde{c}.(Q_1\sigma_1 \mid \dots \mid Q_m\sigma_m)$ such that there is an injective function $inj : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ with $Q_i \in derivatives(P_{S_{inj(i)}})$ and $\sigma_i : fn(Q_i) \rightarrow \tilde{c} \cup fn(P_{\mathcal{FC}})$.

For the derivatives Q of P_S it holds that the identifiers in Q are in the orbit of P_S . Combined with the previous lemma, this relates the identifiers in a reachable fragment and the orbits in the initial process.

Lemma 4. If $Q \in derivatives(P_S)$ then $ident(Q) \subseteq orb(P_S)$.

By an induction along the structure of processes we show that for all $P \in \mathcal{P}$ the following holds: if $Q \in der(P)$ then $ident(Q) \subseteq ident(P)$. With this observation, Lemma 4 follows by an induction on the structure of $derivatives(P_S)$.

We return to the argumentation on Theorem 1. Consider a reachable process $Q \equiv \Pi^k F \mid Q'$ for some Q' . By Lemma 3, $Q \equiv \nu \tilde{c}.(Q_1\sigma_1 \mid \dots \mid Q_m\sigma_m)$ with $Q_i \in derivatives(P_{S_{inj(i)}})$. By transitivity, $\Pi^k F \mid Q' \equiv \nu \tilde{c}.(Q_1\sigma_1 \mid \dots \mid Q_m\sigma_m)$. By Lemmata 1 and 2, $\Pi^k F \mid (Q')_\nu \hat{=} \Pi_{i \in I} G_i = (\nu \tilde{c}.(Q_1\sigma_1 \mid \dots \mid Q_m\sigma_m))_\nu$ for some fragments G_i .

By definition of $\hat{=}$, k of the G_i s are structurally congruent. As identifiers are preserved by $\equiv$, these G_i s have the same identifiers. Each G_i is a parallel composition of some $Q_i\sigma_i$ s. With Lemma 4, $Q_i \in derivatives(P_{S_{inj(i)}})$ implies $ident(Q_i) \subseteq orb(P_{S_{inj(i)}})$. Since every G_i consists of different Q_i s and inj is injective, we have k processes $P_{S_{inj(i)}}$ sharing identifiers, i.e., Theorem 1 holds.

In the case the orbits of all P_{S_i} in $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ are pairwise disjoint, Theorem 1 implies the safeness of the Petri net $\mathcal{PN}[\![P_{\mathcal{FC}}]\!]$. In the following section we show that every FCP can be translated into a bisimilar process with disjoint orbits.

5 From FCPs to Safe Processes

Safe nets are a prerequisite to apply efficient unfolding-based verification techniques. According to Theorem 1, the reason for non-safeness of the nets of arbitrary FCPs is the intersection of orbits. In this section we investigate a translation of FCPs into their syntactic subclass called *safe processes*, where the sequential processes comprising an FCP have pairwise disjoint orbits. The idea of translating $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ to the safe process $\text{Safe}(P_{\mathcal{FC}})$ is to create copies of the process identifiers that are shared among several P_{S_i} , i.e., of those that belong to several orbits. (The corresponding defining equations are duplicated as well.) The intuition is that every P_{S_i} gets its own set of process identifiers (together with the corresponding defining equations) which it can call during system execution. Hence, due to Theorem 1, safe processes are mapped to safe Petri nets.

The main result in this section states that the processes $P_{\mathcal{FC}}$ and $\text{Safe}(P_{\mathcal{FC}})$ are bisimilar, and, moreover, that the fragments are preserved in some sense. Furthermore, the size of the specification $\text{Safe}(P_{\mathcal{FC}})$ is at most quadratic in the size of $P_{\mathcal{FC}}$, and this translation is optimal.

Definition 2. An FCP $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ is a safe process if the orbits of all P_{S_i} are pairwise disjoint, i.e., for all $i, j \in \{1, \dots, n\}$: if $i \neq j$ then $\text{orb}(P_{S_i}) \cap \text{orb}(P_{S_j}) = \emptyset$. $\diamond$

To translate an FCP $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ into a safe process $\text{Safe}(P_{\mathcal{FC}})$, we choose unique numbers for every sequential process, say i for P_{S_i} . We then rename every process identifier K in the orbit of P_{S_i} to a fresh identifier K^i using the unique number i . We use the functions $\text{ren}_k : \mathcal{P} \rightarrow \mathcal{P}$, defined for every $k \in \mathbb{N}$ by

$$\begin{aligned} \text{ren}_k(K) &\stackrel{\text{df}}{=} K^k & \text{ren}_k(K[\tilde{a}]) &\stackrel{\text{df}}{=} \text{ren}_k(K)[\tilde{a}] \\ \text{ren}_k(\sum_{i \in I} \pi_i.P_i) &\stackrel{\text{df}}{=} \sum_{i \in I} \pi_i.\text{ren}_k(P_i) & \text{ren}_k(P \mid Q) &\stackrel{\text{df}}{=} \text{ren}_k(P) \mid \text{ren}_k(Q) \\ \text{ren}_k(\nu a.P) &\stackrel{\text{df}}{=} \nu a.\text{ren}_k(P). \end{aligned}$$

Employing the ren_k function, the FCP $P_{\mathcal{FC}} = \nu\tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ is translated into a safe process as follows:

$$\text{Safe}(P_{\mathcal{FC}}) \stackrel{\text{df}}{=} \nu\tilde{a}.(\text{ren}_1(P_{S_1}) \mid \dots \mid \text{ren}_n(P_{S_n})),$$

where the defining equation of K_S^k is $K_S^k(\tilde{x}) := \text{ren}_k(P_S)$ if $K_S(\tilde{x}) := P_S$. The original defining equations $K_S(\tilde{x}) := P_S$ are then removed. We demonstrate this translation on our running example.

Example 3. Consider the FCP $P_{\mathcal{FC}} = C[\text{url}] \mid C[\text{url}] \mid S[\text{url}]$ in Example 1. The translation is $\text{Safe}(P_{\mathcal{FC}}) = C^1[\text{url}] \mid C^2[\text{url}] \mid S^3[\text{url}]$, where

$$\begin{aligned} C^i(\text{url}) &:= \nu ip.\overline{\text{url}}\langle ip \rangle.ip(s).s(x).C^i[\text{url}], \quad i = 1, 2 \\ S^3(\text{url}) &:= \text{url}(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S^3[\text{url}]. \end{aligned}$$

The equations for C and S are removed. $\diamond$

In the example, we just created another copy of the equation defining a client. In fact, the following result shows that the size of the translated system is at most quadratic in the size of the original specification. We measure the *size* of a π -Calculus process as the sum of the sizes of all the defining equations and the size of the main process. The size of a process is the number of prefixes, operators, and identifiers (with parameters) it uses. So the size of $\mathbf{0}$ is 1, the size of $K[\tilde{a}]$ is $1 + |\tilde{a}|$, the size of $\sum_{i \in I \neq \emptyset} \pi_i.P_i$ is $2|I| - 1 + \sum_{i \in I} \text{size}(P_i)$ (as there are $|I|$ prefixes and $|I| - 1$ pluses), the size of $P \mid Q$ is $1 + \text{size}(P) + \text{size}(Q)$, and the size of $\nu a.P$ is $1 + \text{size}(P)$.

Proposition 1 (Size). *Let $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ be an FCP. Then $\text{size}(\text{Safe}(P_{\mathcal{FC}})) \leq n \cdot \text{size}(P_{\mathcal{FC}})$.*

Note that since $n \leq \text{size}(P_{\mathcal{FC}})$, this result shows that the size of $\text{Safe}(P_{\mathcal{FC}})$ is at most quadratic in the size of $P_{\mathcal{FC}}$.

$\text{Safe}(P_{\mathcal{FC}})$ is a safe process; this follows from the compatibility of the renaming function with the function orb : $\text{orb}(\text{ren}_k(P)) = \text{ren}_k(\text{orb}(P))$.

Proposition 2 (Safeness). *Let $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ be an FCP. Then $\text{Safe}(P_{\mathcal{FC}})$ is a safe process.*

The translation of $P_{\mathcal{FC}}$ into $\text{Safe}(P_{\mathcal{FC}})$ does not alter the behaviour of the process: both processes are bisimilar with a meaningful bisimulation relation. This relation shows that the processes reachable from $P_{\mathcal{FC}}$ and $\text{Safe}(P_{\mathcal{FC}})$ coincide up to the renaming of process identifiers. Thus, not only the behaviour of $P_{\mathcal{FC}}$ is preserved by $\text{Safe}(P_{\mathcal{FC}})$, but also the structure of the reachable process terms, in particular their fragments. Technically, we define the relation $\mathcal{R}_i$ by $(P, Q) \in \mathcal{R}_i$ iff there are names $\tilde{a}$ and sequential processes $P_{S_1}, \dots, P_{S_n}$, where the topmost operator of every P_{S_i} is different from ν , such that

$$P \equiv \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n}) \text{ and } Q \equiv \nu \tilde{a}.(P_{S_1} \mid \dots \mid \text{ren}_i(P_{S_i}) \mid \dots \mid P_{S_n}).$$

Note that it is obvious from this definition that $\nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ and $\nu \tilde{a}.(P_{S_1} \mid \dots \mid \text{ren}_i(P_{S_i}) \mid \dots \mid P_{S_n})$ are related by $\mathcal{R}_i$.

Theorem 2. *For any $i \in \{1, \dots, n\}$, the relation $\mathcal{R}_i$ is a bisimulation that relates the FCPs $\nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ and $\nu \tilde{a}.(P_{S_1} \mid \dots \mid \text{ren}_i(P_{S_i}) \mid \dots \mid P_{S_n})$.*

By transitivity of bisimulation, Theorem 2 allows for renaming several P_{S_i} and still gaining a bisimilar process. In particular, renaming all n processes in $P_{\mathcal{FC}} = \nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ yields the result for the safe system $\text{Safe}(P_{\mathcal{FC}})$.

Consider a process Q which is reachable from $P_{\mathcal{FC}}$. We argue that the structure of Q is essentially preserved (1) by the translation of $P_{\mathcal{FC}}$ to the safe process $\text{Safe}(P_{\mathcal{FC}})$ and then (2) by the translation of $\text{Safe}(P_{\mathcal{FC}})$ to the safe Petri net $\mathcal{PN}[\text{Safe}(P_{\mathcal{FC}})]$. With this result we can reason about the *structure of all processes reachable from $P_{\mathcal{FC}}$* using $\mathcal{PN}[\text{Safe}(P_{\mathcal{FC}})]$.

According to Theorem 2, $P_{\mathcal{FC}}$ and $\text{Safe}(P_{\mathcal{FC}})$ are bisimilar via the relation $\mathcal{R}_1 \circ \dots \circ \mathcal{R}_n$, e.g., a process $Q = \nu a. \nu b. (H[a] \mid K[a] \mid L[b])$ reachable from $P_{\mathcal{FC}}$ corresponds to $Q' = \nu a. \nu b. (H^1[a] \mid K^2[a] \mid L^3[b])$ reachable from $\text{Safe}(P_{\mathcal{FC}})$. Hence, one can reconstruct the fragments of Q from those of Q' .

Indeed, compute the restricted forms: $(Q)_\nu = \nu a.(H[a] \mid K[a]) \mid \nu b.L[b]$ and $(Q')_\nu = \nu a.(H^1[a] \mid K^2[a]) \mid \nu b.L^3[b]$. Dropping the superscripts in $(Q')_\nu$ yields the fragments in $(Q)_\nu$, since only the restricted names influence the restricted form of a process, not the process identifiers. The transition systems of $\text{Safe}(P_{\mathcal{FC}})$ and $\mathcal{PN}[\llbracket \text{Safe}(P_{\mathcal{FC}}) \rrbracket]$ are isomorphic, e.g., Q' corresponds to the marking $M = \{[\nu a.(H^1[a] \mid K^2[a])], [\nu b.L^3[b]]\}$ [Mey07, Theorem 1]. Thus, from a marking of $\mathcal{PN}[\llbracket \text{Safe}(P_{\mathcal{FC}}) \rrbracket]$ one can obtain the restricted form of a reachable process in $\text{Safe}(P_{\mathcal{FC}})$, which in turn corresponds to the restricted form in $P_{\mathcal{FC}}$ (when the superscripts of process identifiers are dropped). Furthermore, the bisimulation between $P_{\mathcal{FC}}$ and $\mathcal{PN}[\llbracket \text{Safe}(P_{\mathcal{FC}}) \rrbracket]$ allows one to reason about the *behaviour* of $P_{\mathcal{FC}}$ using $\mathcal{PN}[\llbracket \text{Safe}(P_{\mathcal{FC}}) \rrbracket]$. (This bisimulation follows from the bisimulation between $P_{\mathcal{FC}}$ and $\text{Safe}(P_{\mathcal{FC}})$ and the isomorphism of the transition systems of $\text{Safe}(P_{\mathcal{FC}})$ and $\mathcal{PN}[\llbracket \text{Safe}(P_{\mathcal{FC}}) \rrbracket]$).

We discuss our choice to rename all P_{S_i} in $\nu \tilde{a}.(P_{S_1} \mid \dots \mid P_{S_n})$ to gain a safe process. One might be tempted to improve our translation by renaming only a subset of processes P_{S_i} whose orbits intersect with many others, in hope to get a smaller specification than $\text{Safe}(P_{\mathcal{FC}})$. We show that this idea does not work, and the resulting specification will be of the same size, i.e., our definition of $\text{Safe}(P_{\mathcal{FC}})$ is *optimal*. First, we illustrate this issue with an example.

Example 4. Let $P = \tau.K[\tilde{a}] + \tau.L[\tilde{a}]$, $R = \tau.K[\tilde{a}]$ and $S = \tau.L[\tilde{a}]$, where $K(\tilde{x}) := \text{Def}_1$ and $L(\tilde{x}) := \text{Def}_2$. Consider the process $P \mid R \mid S$. The orbits of P and R as well as the orbits of P and S intersect.

The renaming of P yields $\text{ren}_1(P) \mid R \mid S = \tau.K^1[\tilde{a}] + \tau.L^1[\tilde{a}] \mid R \mid S$, where K and L are defined above and $K^1(\tilde{x}) := \text{ren}_1(\text{Def}_1)$, $L^1(\tilde{x}) := \text{ren}_1(\text{Def}_2)$. This means we create additional copies of the shared identifiers K and L .

The renaming of R and S yields the process $P \mid \text{ren}_1(R) \mid \text{ren}_2(S) = P \mid \tau.K^1[\tilde{a}] \mid \tau.L^2[\tilde{a}]$, where we create new defining equations for the identifiers K^1 and L^2 . The size of our translation is the same. $\diamond$

This illustrates that any renaming of processes P_{S_i} where the orbits overlap results in a specification of the same size. To render this intuition precisely, we call $K_S^k(\tilde{x}) := \text{ren}_k(P_S)$ a *copy of the equation* $K_S(\tilde{x}) := P_S$, for any $k \in \mathbb{N}$. We also count $K_S(\tilde{x}) := P_S$ as a copy of itself.

Proposition 3 (Necessary condition for safeness). *The number of copies of an equation $K_S(\tilde{x}) := P_S$ necessary to get a safe process from $P_{\mathcal{FC}}$ equals to the number of orbits that contain K_S .*

Now we show that our translation provides precisely this minimal number of copies of defining equations for every identifier, i.e., that it is optimal.

Proposition 4 (Optimality of our translation). *Our translation $\text{Safe}(P_{\mathcal{FC}})$ provides as many copies of a defining equation $K_S(\tilde{x}) := P_S$ as there are orbits containing K_S .*

Remark 1. Note that one can, in general, optimise the translation by performing some dynamic (rather than syntactic) analysis, and produce a smaller process

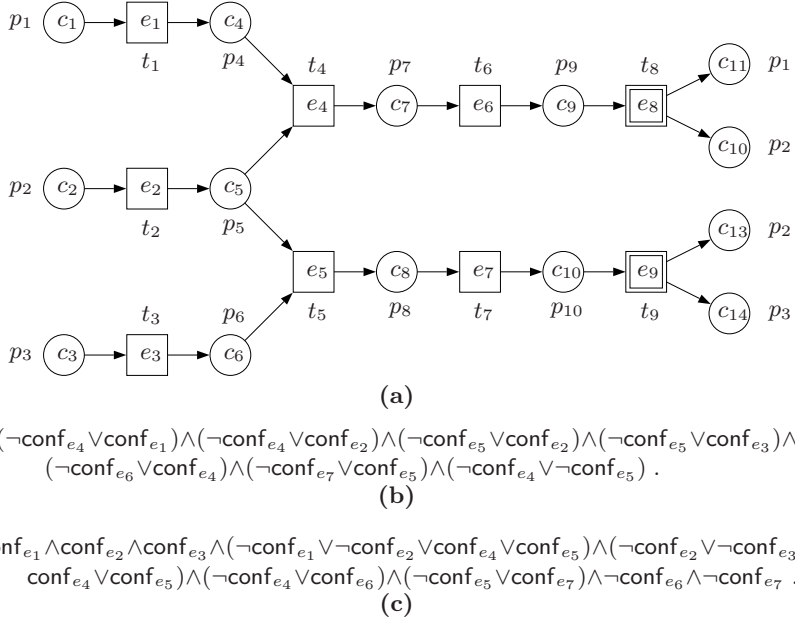


Fig. 2. A finite and complete unfolding prefix of the Petri net in Figure 1(b) (a), the corresponding configuration constraint CONF (b), and the corresponding violation constraint VIOL expressing the deadlock condition (c).

whose corresponding Petri net is safe; however, our notion of a safe process is syntactic rather than dynamic, and so the resulting process will not be safe according to our definition. $\diamond$

6 Net Unfoldings

A *finite and complete unfolding prefix* of a bounded Petri net $\mathcal{T}$ is a finite acyclic net which implicitly represents all the reachable states of $\mathcal{T}$ together with transitions enabled at those states. Intuitively, it can be obtained through *unfolding* $\mathcal{T}$, by successive firing of transitions, under the following assumptions: (i) for each new firing a fresh transition (called an *event*) is generated; (ii) for each newly produced token a fresh place (called a *condition*) is generated. For example, a finite and complete prefix of the Petri net in Figure 1(b) is shown in Figure 2(a). Due to its structural properties (such as acyclicity), the reachable states of $\mathcal{T}$ can be represented using *configurations* of its unfolding. A configuration C is a downward-closed set of events (being downward-closed means that if $e \in C$ and f is a causal predecessor of e then $f \in C$) without *choices* (i.e., for all distinct events $e, f \in C$, $\bullet e \cap \bullet f = \emptyset$). For example, in the prefix in Figure 2(a), $\{e_1, e_2, e_4\}$ is a configuration, whereas $\{e_1, e_2, e_6\}$ and $\{e_1, e_2, e_3, e_4, e_5\}$ are not (the former does not include e_4 , which is a predecessor of e_6 , while the latter

contains a choice between e_4 and e_5). Intuitively, a configuration is a partially ordered execution, i.e., an execution where the order of firing some of its events (viz. concurrent ones) is not important; e.g., the configuration $\{e_1, e_2, e_4\}$ corresponds to two totally ordered executions reaching the same *final* marking: $e_1e_2e_4$ and $e_2e_1e_4$. Since a configuration can correspond to multiple executions, it is often much more efficient in model checking to explore configurations rather than executions. We will denote by $[e]$ the *local* configuration of an event e , i.e., the smallest (w.r.t. $\subseteq$) configuration containing e (it is comprised of e and its causal predecessors).

The unfolding is infinite whenever the original $\mathcal{Y}$ has an infinite run; however, since $\mathcal{Y}$ is bounded and hence has only finitely many reachable states, the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events) without loss of information, yielding a finite and complete prefix. Intuitively, an event e can be declared cut-off if the already built part of the prefix contains a configuration C^e (called the *corresponding* configuration of e) such that its final marking coincides with that of $[e]$ and C^e is smaller than $[e]$ w.r.t. some well-founded partial order on the configurations of the unfolding, called an *adequate order* [ERV02].

Efficient algorithms exist for building such prefixes [ERV02, Kho03], which ensure that the number of non-cut-off events in a complete prefix never exceeds the number of reachable states of the original Petri net. Moreover, complete prefixes are often exponentially smaller than the corresponding state graphs, especially for highly concurrent Petri nets, because they represent concurrency directly rather than by multidimensional interleaving ‘diamonds’ as it is done in state graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the state graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will coincide with the net itself. Also, if the example in Figure 1(b) is scaled up (by increasing the number of clients), the size of the prefix is linear in the number of clients, even though the number of reachable states grows exponentially. Thus, unfolding prefixes significantly alleviate the state explosion in many practical cases.

A fundamental property of a finite and complete prefix is that each reachable marking of $\mathcal{Y}$ is a final marking of some configuration C (without cut-offs) of the prefix, and, conversely, the final marking of each configuration C of the prefix is a reachable marking in $\mathcal{Y}$. Thus various reachability properties of $\mathcal{Y}$ (e.g., marking and sub-marking reachability, fireability of a transition, mutual exclusion of a set of places, deadlock, and many others) can be restated as the corresponding properties of the prefix, and then checked, often much more efficiently.

Most of ‘interesting’ computation problems for safe Petri nets are PSPACE-complete [Esp98], but the same problems for prefixes are often in NP or even P. (Though the size of a finite and complete unfolding prefix can be exponential in the size of the original Petri net, in practice it is often relatively small, as explained above.) A reachability property of $\mathcal{Y}$ can easily be reformulated for a prefix, and then translated into some canonical problem, e.g., Boolean satisfiability (SAT). Then an off-the-shelf solver can be used for efficiently solving

it. Such a combination ‘unfolder & solver’ turns out to be quite powerful in practice [KKY04].

Unfolding-Based Model Checking. This paper concentrates on the following approach to model checking. First, a finite and complete prefix of the Petri net unfolding is built. It is then used for constructing a Boolean formula encoding the model checking problem at hand. (It is assumed that the property being checked is the unreachability of some ‘bad’ states, e.g., deadlocks.) This formula is unsatisfiable iff the property holds, and such that any satisfying assignment to its variables yields a trace violating the property being checked.

Typically such a formula would have for each non-cut-off event e of the prefix a variable conf_e (the formula might also contain other variables). For every satisfying assignment A , the set of events $C \stackrel{\text{df}}{=} \{e \mid \text{conf}_e = 1\}$ is a configuration whose final marking violates the property being checked. The formula often has the form $\text{CONF} \wedge \text{VIOL}$. The role of the property-independent *configuration constraint* CONF is to ensure that C is a configuration of the prefix (not just an arbitrary set of events). CONF can be defined as the conjunction of the formulae

$$\bigwedge_{e \in E \setminus E_{\text{cut}}} \bigwedge_{f \in \bullet\bullet e} (\neg \text{conf}_e \vee \text{conf}_f) \quad \text{and} \quad \bigwedge_{e \in E \setminus E_{\text{cut}}} \bigwedge_{f \in Ch_e} (\neg \text{conf}_e \vee \neg \text{conf}_f),$$

where $Ch_e \stackrel{\text{df}}{=} \{((\bullet e) \bullet \setminus \{e\}) \setminus E_{\text{cut}}\}$ is the set of non-cut-off events which are in the direct choice relation with e . The former formula is basically a set of implications ensuring that if $e \in C$ then its immediate predecessors are also in C , i.e., C is downward closed. The latter one ensures that C contains no choices. CONF is given in the *conjunctive normal form (CNF)* as required by most SAT solvers. For example, the configuration constraint for the prefix in Figure 2(a) is shown in part (b) of this figure. The size of this formula is quadratic in the size of the prefix, but can be reduced down to linear by introducing auxiliary variables.

The role of the property-dependent *violation constraint* VIOL is to express the property violation condition for a configuration C , so that if a configuration C satisfying this constraint is found then the property does not hold, and C can be translated into a violation trace. For example, for the deadlock condition VIOL can be defined as

$$\bigwedge_{e \in E} \left(\bigvee_{f \in \bullet\bullet e} \neg \text{conf}_f \vee \bigvee_{f \in (\bullet e) \bullet \setminus E_{\text{cut}}} \text{conf}_f \right).$$

This formula requires for each event e (including cut-off events) that some event in $\bullet\bullet e$ has not fired or some of the non-cut-off events (including e unless it is cut-off) consuming tokens from $\bullet e$ has fired, and thus e is not enabled. This formula is given in the CNF. For example, the violation constraint for the deadlock checking problem formulated for the prefix in Figure 2(a) is shown in part (c) of this figure. The size of this formula is linear in the size of the prefix.

If VIOL is a formula of polynomial size (in the size of the prefix) then one can check $\text{CONF} \wedge \text{VIOL}$ for satisfiability in non-deterministic polynomial time. In particular, every polynomial size (w.r.t. the prefix) formula F over the places of the net can be translated into a VIOL formula that is polynomial in the

size of the prefix. Here, an atomic proposition p of F holds iff place p carries a token (we deal with safe nets). This covers reachability of markings and submarkings, deadlocks, mutual exclusion, and many other properties. Furthermore, an unfolding technique for model checking state-based LTL-X is presented in [EH01]. State-based means that the atomic propositions in the logic are again the places of the Petri net.

7 Experimental Results

To demonstrate the practicality of our approach, we implemented the translation of π -Calculus to Petri nets discussed in Section 3 and the translation of FCPs to safe processes presented in Section 5. In this section, we apply our tool chain to check three series of benchmarks for deadlocks. We compare the results with other well-known approaches and tools for π -Calculus verification.

The *NESS* (*Newcastle E-Learning Support System*) example models an electronic coursework submission system. This series of benchmarks is taken from [KKN06], where the only other unfolding-based verification technique for the π -Calculus is presented. The approach described in [KKN06] is limited to the *finite* π -Calculus, a subset of π -Calculus allowing to express only finite behaviours. It translates finite π -Calculus terms into high-level Petri nets and model checks the latter. The translation into Petri nets used in [KKN06] is very different from our approach, and a high-level net unfoldier is used there for verification, while our technique uses the standard unfolding procedure for safe low-level nets. Moreover, our technique is not limited to the finite π -Calculus.

The model consists of a teacher process T composed in parallel with k students S (the system can be scaled up by increasing the number of students) and an environment process ENV . Every student has its own local channel for communication, h_i , and all students share the channel h :

$$\nu h. \nu h_1. \dots \nu h_k. (T[nesc, h_1, \dots, h_k] \mid \Pi_{i=1}^k S[h, h_i] \mid ENV[nesc]) .$$

The idea is that the students are supposed to submit their work for assessment to *NESS*. The teacher passes the channel *nesc* of the system to all students, $\overline{h_i}\langle nesc \rangle$, and then waits for the confirmation that they have finished working on the assignment, $h_i(x)$. After receiving the nesc channel, $h_i(nsc)$, students organise themselves in pairs. To do so, they send their local channel h_i on h and at the same time listen on h to receive a partner, $\overline{h}\langle h_i \rangle \dots + h(x) \dots$. When they finish, exactly one student of each pair sends two channels to the support system, $\overline{nsc}\langle h_i \rangle. \overline{nsc}\langle x \rangle$, which give access to their completed joint work. These channels are received by the *ENV* process. The students finally notify the teacher about the completion of their work, $\overline{h_i}\langle fin \rangle$. Thus, the system is modelled by:

$$\begin{aligned} T(nesc, h_1, \dots, h_k) &:= \Pi_{i=1}^k \overline{h_i}\langle nesc \rangle. h_i(x_i). \mathbf{0} \\ S(h, h_i) &:= h_i(nsc). (\overline{h}\langle h_i \rangle. \overline{h_i}\langle fin \rangle. \mathbf{0} + h(x). \overline{nsc}\langle h_i \rangle. \overline{nsc}\langle x \rangle. \overline{h_i}\langle fin \rangle. \mathbf{0}) \\ ENV(nesc) &:= nesc(y_1). \dots . nesc(y_k). \mathbf{0} \end{aligned}$$

Table 1. Experimental results I

Mod.	FCP Size	HLNet		Model Checking					mwb	hal π 2fc	Struct			Safe Size	Struct			Model Checking				
		P	T	unf	B	E*	sat	P			T	B	P		T	unf	B	E*	sat			
dns4	84	1433	511	6	10429	181	< 1	10	93	22	47	8	98	32	50	< 1	113	38	< 1			
dns6	123	3083	1257	46	28166	342	< 1	-	-	32	94	12	145	48	99	< 1	632	159	< 1			
dns8	162	5357	2475	354	58863	551	< 1	-	-	42	157	16	192	64	164	< 1	3763	745	< 1			
dns10	201	8255	4273	-	-	-	-	-	-	52	236	20	271	80	239	1	22202	3656	2			
dns12	240	11777	6791	-	-	-	-	-	-	62	331	24	324	96	286	56	128295	18192	62			
ns2	61	157	200	1	5553	127	< 1	< 1	< 1	18	28	4	67	26	40	< 1	61	27	< 1			
ns3	88	319	415	7	22222	366	< 1	1	8	37	91	6	98	56	141	< 1	446	153	< 1			
ns4	115	537	724	69	101005	1299	1	577	382	68	229	8	129	102	364	< 1	5480	1656	< 1			
ns5	142	811	1139	532	388818	4078	58	-	-	119	511	10	160	172	815	17	36865	7832	3			
ns6	169	1141	1672	-	-	-	-	-	-	206	1087	12	191	282	1722	1518	-	377920	65008	84		
ns7	196	1527	2335	-	-	-	-	-	-	361	2297	14	222	646	3605	-	-	-	-	-		
ns2-r	61							n/a	n/a	16	24	4	67	24	36	< 1	51	22	< 1			
ns3-r	88							n/a	n/a	29	70	6	98	48	117	< 1	292	99	< 1			
ns4-r	113							n/a	n/a	45	123	8	127	79	216	< 1	1257	392	< 1			
ns5-r	140							n/a	n/a	66	241	10	158	119	435	2	10890	2635	1			
ns6-r	167							n/a	n/a	91	418	12	189	167	768	123	-	107507	19892	31		
ns7-r	194							n/a	n/a	120	666	14	220	223	1239	-	-	-	-	-		

In the following Table 1, the row **nsk** gives the verification results for the *NESS* system with $k \in \mathbb{N}$ students. The property we verified was whether all processes successfully terminate by reaching the end of their individual code (as distinguished from a deadlock where some processes are stuck in the middle of their intended behaviour, waiting for a communication to occur). Obviously, the system successfully terminates iff the number of students is even, i.e., they can be organised into pairs. The **dnsk** entries refer to a refined *NESS* model where the pairing of students is deterministic; thus the number of students is even, and these benchmarks are deadlock-free.

The second example is the client-server system similar to our running example. For a more realistic model, we extend the server to spawn separate sessions that handle the clients' requests. We change the server process in Section 2 to a more concurrent *CONCS* and add separate session processes:

$$\begin{aligned}
CONCS(url, getses) &:= url(y).getses(s).\overline{y}\langle s \rangle.CONCS[url, getses] \\
SES(getses) &:= \nu ses.\overline{getses}\langle ses \rangle.\overline{ses}\langle ses \rangle.SES[getses]
\end{aligned}$$

On a client's request, the server creates a new session object using the *getses* channel, *getses*(*s*). A session object is modelled by a *SES* process. It sends its private channel νses along the *getses* channel to the server. The server forwards the session to the client, $\overline{y}\langle s \rangle$, which establishes the private session, and becomes available for further requests. This case study uses recursion and is scalable in the number of clients and the number of sessions. In Table 2, e.g., the entry 5s5c gives the verification results for the system with five *SES* processes, five *C* processes and one server. All these benchmarks are deadlock-free.

The last example is the well-known specification of the handover procedure in the GSM Public Land Mobile Network. We use the standard π -Calculus model with one mobile station, two base stations, and one mobile switching center presented in [OP92].

We compare our results with three other techniques for π -Calculus verification: the mentioned approach in [KKN06], the verification kit *HAL* [FGMP03], and the *mobility workbench* (*MWB*) [VM94]. *HAL* translates a π -Calculus process into a history dependent automaton (HDA) [Pis99]. This in turn is

Table 2. Experimental results II

Model	FCP	mwb	hal	Struct			Safe	Struct			Model Checking			
	Size	dl	π 2fc	P	T	B	Size	P	T	unf	B	E*	sat	
gsm	194	-	18	374	138	1	194	148	344	< 1	345	147	< 1	
gsm-r	194	n/a	n/a	60	72	1	194	75	110	< 1	150	72	< 1	
1s1c	44	-	< 1	11	13	1	44	12	15	< 1	17	9	< 1	
1s2c	47	-	6	12	15	2	58	22	30	< 1	35	17	< 1	
2s1c	47	-	2	20	31	2	56	22	35	< 1	37	18	< 1	
2s2c	50	-	138	31	59	2	70	40	66	< 1	73	33	< 1	
3s2c	53	-	-	68	159	3	82	66	128	< 1	137	57	< 1	
3s3c	56	-	-	85	217	3	96	100	194	< 1	216	87	< 1	
4s4c	63	-	-	362	1202	4	122	216	484	< 1	537	195	< 1	
5s5c	68	-	-	980	3818	5	148	434	1132	< 1	1238	403	< 1	

translated into a finite automaton which is checked using standard tools. The *MWB* does not use any automata translation, but builds the state space on the fly. These tools can verify various properties, but we perform our experiments for deadlock checking only, as it is the common denominator of all these tools.

We briefly comment on the role of the models with the suffix *-r* in Table 1. One can observe that parallel compositions inside a fragment lead to interleaving ‘diamonds’ in our Petri net representation. Thus, restricted names that are known to a large number of processes can make the size of our Petri net translation grow exponentially. We demonstrate this effect by verifying some of the *NESS* benchmarks with and without (suffix *-r* in the table) the restrictions on such critical names. Even with the critical restrictions our approach outperforms the other tools; but when such restrictions are removed, it becomes orders of magnitude faster. (Removing such critical restrictions does not alter the process behaviour: $\nu a.P$ can evolve into $\nu a.P'$ iff P can evolve into P' ; thus, one can replace $\nu a.P$ by P for model checking purposes.)

The columns in Tables 1 and 2 are organised as follows. **FCP Size** gives the size of the process as defined in Section 5. The following two columns, **HLNet** and **Model Checking** (present only in Table 1), are the verification results when the approach in [KKN06] is applied. In the former column, $|P|$ and $|T|$ state the number of places and transitions in the high-level Petri net. The following column **unf** gives the time to compute the unfolding prefix of this net. (We measure all runtimes in seconds.) For this prefix, $|B|$ is the number of conditions, and $|E^*|$ is the number of events (excluding cut-offs). Like our technique, the [KKN06] employs a SAT solver whose runtime is given by **sat**. The following two columns, **mwb dl** and **hal $\pi 2fc$** , give the runtimes for the deadlock checking algorithm in *MWB* and for converting a π -Calculus process into a finite automaton (via HDA). This time includes the translation of a π -Calculus process into an HDA, minimisation of this HDA, and the conversion of the minimised HDA into a finite automaton [FGMP03]. The remaining entries are the results of applying our model checking procedure. The column **Struct** gives the numbers of places and transitions and the bounds of the Petri nets corresponding to a direct translation of the FCPs. These nets are given only for comparison, and are not used for model checking. **Safe Size** gives the size of the safe process computed by the function *Safe* in Section 5, and the next column gives the numbers of places and transitions of the corresponding safe Petri nets. Note that these nets, unlike those in [KKN06], are the usual low-level Petri nets. The following columns give

the unfolding times, the prefix sizes, and the times for checking deadlocks on the prefixes using a SAT solver. A ‘—’ in the tables indicates the corresponding tool did not produce an output within 30 minutes, and an ‘n/a’ means the technique was not applicable to the example.

Table 1 illustrates the results for checking the *NESS* example with the different techniques. As the *MWB* requires processes where all names are restricted, we cannot check the $-r$ versions of the case studies. Our runtimes are by orders of magnitude smaller in comparison with *HAL* and *MWB*, and are much better compared with the approach in [KKN06]. Furthermore, they dramatically improve when the critical names are removed (the $-r$ models).

The approach in [KKN06] only applies to the finite π -Calculus, so one cannot check the client-server or the GSM benchmarks with that technique. Table 2 shows that the proposed technique dramatically outperforms both *MWB* and *HAL*, and handles the benchmark with five sessions and clients within a second.

8 Conclusions and Future Work

In this paper, we have proposed a practical approach for verification of finite control processes. It works by first translating the given FCP into a safe process, and then translating the latter into a safe Petri net for which unfolding-based model checking is performed. Our translation to safe processes exploits a general boundedness result for FCP nets based on the theory of orbits. Our experiments show that this approach has significant advantages over other existing tools for verification of mobile systems in terms of memory consumption and runtime. We plan to further develop this approach, and below we identify potential directions for future research.

It would be useful to extend some temporal logic so that it could express interesting properties of π -Calculus. (The usual LTL does not capture, at least in a natural way, the communication in dynamically created channels and dynamic connectivity properties.) Due to our fragment-preserving (i.e., preserving the local connections of processes) bisimulation result, one should be able to translate such properties into Petri net properties for verification. The hope is that since this Petri net has a rich structure (e.g., the connectivity information can be retrieved from place annotations), the resulting properties can be expressed in some standard logic such as state-based LTL and then efficiently model checked with existing techniques.

One can observe that after the translation into a safe process, some fragments differ only by the replicated process identifiers. Such fragments are equivalent in the sense that they react in the same way and generate equivalent places in the postsets of the transitions. Hence, it should be possible to optimise our translation procedure, because many structural congruence checks can be omitted and several computations of enabled reactions become unnecessary. Moreover, this observation allows one to use in the unfolding procedure a weaker (compared with the equality of final markings) equivalence on configurations, as explained

in [Kho03, Section 2.5]. This would produce cut-off events more often and hence reduce the size of the unfolding prefix.

It seems to be possible to generalise our translation to a wider subclass of π -Calculus. For example, consider the process $S[url] \mid C[url] \mid C[url]$ modelling a *concurrent* server and two clients, with the corresponding process identifiers defined as

$$\begin{aligned} S(url) &:= url(y).(\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle).\mathbf{0} \mid S[url] \\ C(url) &:= \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C[url] \end{aligned}$$

Intuitively, when contacted by a client, the server spawns a new session and is ready to serve another client, i.e., several clients can be served in parallel. Though this specification is not an FCP, it still results in a 2-bounded Petri net very similar to the one in Figure 1(a). Our method can still be used to convert it into a safe Petri net for subsequent verification.

Acknowledgements. The authors would like to thank Maciej Koutny, Eike Best and Ernst-Rüdiger Olderog for helpful comments and for arranging the visit of the first author to Newcastle.

This research was supported by the German Research Council (DFG) as part of the Graduate School GRK 1076/1 (TRUSTSOFT) and by the Royal Academy of Engineering/EPSRC post-doctoral research fellowship EP/C53400X/1 (DAVAC).

References

- [CGP99] Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
- [Dam96] Dam, M.: Model checking mobile processes. *Information and Computation* 129(1), 35–51 (1996)
- [EH01] Esparza, J., Heljanko, K.: Implementing LTL model checking with net unfoldings. In: Dwyer, M.B. (ed.) *SPIN 2001*. LNCS, vol. 2057, pp. 37–56. Springer, Heidelberg (2001)
- [ERV02] Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan’s Unfolding Algorithm. *Formal Methods in System Design* 20(3), 285–310 (2002)
- [Esp98] Esparza, J.: Decidability and complexity of Petri net problems — an introduction. In: Reisig, W., Rozenberg, G. (eds.) *APN 1998*. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
- [FGMP03] Ferrari, G.-L., Gnesi, S., Montanari, U., Pistore, M.: A model-checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology* 12(4), 440–473 (2003)
- [Kho03] Khomenko, V.: Model Checking Based on Prefixes of Petri Net Unfoldings. PhD thesis, School of Computing Science, Newcastle University (2003)
- [KKN06] Khomenko, V., Koutny, M., Niaouris, A.: Applying Petri net unfoldings for verification of mobile systems. In: *Proc. Workshop on Modelling of Objects, Components and Agents (MOCA 2006)*, Bericht FBI-HH-B-267/06, pp. 161–178. University of Hamburg (2006)
- [KKY04] Khomenko, V., Koutny, M., Yakovlev, A.: Detecting state coding conflicts in STG unfoldings using SAT. *Fundamenta Informaticae* 62(2), 1–21 (2004)

- [McM92] McMillan, K.: Using unfoldings to avoid state explosion problem in the verification of asynchronous circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–174. Springer, Heidelberg (1993)
- [Mey07] Meyer, R.: A theory of structural stationarity in the π -Calculus (under revision) (2007)
- [Mil99] Milner, R.: Communicating and Mobile Systems: the π -Calculus. Cambridge University Press, Cambridge (1999)
- [MKS08] Meyer, R., Khomenko, V., Strazny, T.: A practical approach to verification of mobile systems using net unfoldings. Technical Report CS-TR-1064, School of Computing Science, Newcastle University (2008), URL: <http://www.cs.ncl.ac.uk/research/pubs/trs/abstract.php?number=1064>
- [OP92] Orava, F., Parrow, J.: An algebraic verification of a mobile network. *Formal Aspects of Computing* 4(6), 497–543 (1992)
- [Pis99] Pistore, M.: History Dependent Automata. PhD thesis, Dipartimento di Informatica, Università di Pisa (1999)
- [SW01] Sangiorgi, D., Walker, D.: The π -Calculus: a Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
- [Val98] Valmari, A.: Lectures on Petri Nets I: Basic Models. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998)
- [VM94] Victor, B., Moller, F.: The mobility workbench: A tool for the π -Calculus. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 428–440. Springer, Heidelberg (1994)

Cooperative Arrival Management in Air Traffic Control - A Coloured Petri Net Model of Sequence Planning

Hendrik Oberheid¹ and Dirk Söffker²

¹ German Aerospace Center (DLR), Institute of Flight Guidance,
Braunschweig, Germany

² University of Duisburg-Essen, Chair of Dynamics and Control,
Duisburg, Germany

Abstract. A Coloured Petri Net model implemented in CPN Tools is presented which simulates a potential future arrival planning process in air traffic control. The planning process includes a cooperation between airborne and ground side in which the aircraft involved provide information e.g. with regard to their estimated earliest and latest times of arrival at the airport. This information is then used by a planning system on the ground to establish a favorable sequence in which aircraft will be led to the runway. The model has been built in order to acquire a better understanding of how the behavior of individual actors (i.e. aircraft) within the cooperation influences the outcome of the overall sequence planning process. A peculiarity of the CP-net from a modeling point of view lies in the fact that state space analysis is used repeatedly during each cycle of the planning process to generate and evaluate the potential solutions to the sequence planning problem. The results gained through queries on the state space are then re-fed into the simulation and analysis for the next planning cycle. The results from the model will in future be used to build realistic scenarios and assumptions on how different actors will interact with the system from a human factors point of view.

Keywords: air traffic management, arrival management, sequence planning, coloured Petri nets, state space analysis.

1 Introduction

One of the most challenging tasks of air traffic control consists in managing the arrival traffic at a highly frequented airport. For an increasing number of large airports this task is therefore supported by automated controller assistance tools, so called AMANs (Arrival Managers) [1]. Depending on the capabilities of the specific AMAN the tool will nowadays support the human controller in one or both of the following tasks: *1. sequencing*, i.e. establishing a favorable arrival sequence (sequence of aircraft) according to a number of optimization criteria *2. metering* that is to calculate for each individual aircraft the target times over

(TTO) certain geographic points of the respective arrival route appropriate for the aircraft's position in the sequence and the runway assigned.

Apart from sequencing and metering future AMANs will additionally be able to support the controller in finding an efficient and conflict free route for each aircraft from its current position to the runway threshold (*trajectory generation*) [2]. The AMAN could also generate guidance instructions to the controller with regard to individual clearances¹ he should give to an aircraft in order to lead the aircraft along that route as planned (*clearance generation*). A simplified view of the arrival planning process including the main processes and information flow is illustrated in Fig. 1.

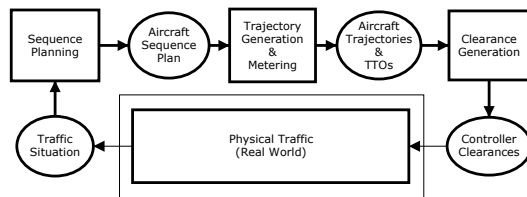


Fig. 1. Simplified view of arrival planning process

A critical determinant for the quality of the arrival planning process is generally the type and accuracy of information on which the planning is based. Most AMANs currently in operation rely on a combination of radar data and flight plan information as their main input to the planning process. As an enhancement to that the DLR Institute of Flight Guidance is developing an Arrival Manager named 4D-CARMA (4D Cooperative Arrival Manager) [3,4] which makes use of data received directly via datalink from individual aircraft to feed the planning process. This data includes components such as estimated earliest and latest times over certain geographic points as well as information on the planned descent profile and speed which are computed by the aircraft's own flight management system.

The tighter cooperation of airborne and groundside envisioned with 4D-CARMA and the higher accuracy of data received directly from the airborne side is expected to increase both the capacity of the airspace, the economic efficiency (e.g. lower fuel consumption) of individual flights, and facilitate environmental improvements with regard to noise abatement. It is also expected that greater flexibility in the planning process will be reached and chances will improve to deliver *user-preferred trajectories* (i.e. "aircraft/airline-preferred trajectories") and accommodate airlines' wishes with regard to the prioritization of specific flights in the sequence. However, as the enhanced system will base its planning on information received from external actors it is important to develop

¹ A *clearance* is an authorization or permission given by air traffic control for an aircraft to proceed along a certain route segment or to execute some other action specified in the body of the clearance.

a thorough understanding on how each piece of information submitted affects the overall system behavior. Not only is it necessary to consider the consequences on the overall outcome arising from submitting certain data, it is also essential to consider the incentives the system provides for each individual actor to fully comprehend his role and behavior while interacting with the system. The CPN model of cooperative arrival planning presented in this paper and the analysis and visualization functions developed in this context are designed to contribute to acquiring the necessary understanding.

Petri Nets and other formal techniques have been successfully applied to the analysis of air traffic control concepts and avionics systems before. Notably, NASA (US National Aeronautics and Space Administration) runs a dedicated formal methods research group [5] in order to support the traditional system validation methods applied in the ATM domain such as testing, flight experiments and human in the loop (HIL) simulations with rigorous mathematical methods.

From the application point of view the systems and concepts which have been analyzed in literature comprise such diverse themes as conflict resolution and recovery methods for aircraft encounters [6], verification of novel arrival and departure procedures inside SCA (Self Controlled Areas) at small regional airports [7], and the analysis of pilot interaction with flight control systems with regard to mode confusion potential [8]. In the majority of cases a strong focus has been laid on the analysis and proof of systems' safety properties [6,7,9,8]. In fewer instances the objective has also been to analyze and improve system efficiency and optimize performance (e.g. [10,11]). With regard to tools and modeling techniques NASA's work [6,7,9] relies largely on the specification and verification system PVS (Prototype Verification system). Examples of Petri net modeling of ATM applications can be found in [11] which uses Design/CPN and in [10] uses CPN Tools as is done in this work.

This work for the first time presents an approach to the formal modeling of cooperative arrival management processes, thus it substantially differs from the above work in particular with regard to the examined ATM application. Coloured Petri Nets are chosen as a the modelling approach here because they offer an intuitive means to modeling and analyzing the combinatorial aspects of the sequence planning problem. A novel method is presented here to handle with CPN Tools [12] the iterative calculation of the large state spaces caused by the application. From an application point of view our current interest is mainly in investigating different dynamic properties of the planning system in order to better understand the incentives the system sets for the different actors. In a further step, this understanding is considered vital to achieve optimal overall system performance and system safety in the future.

2 Sequence Planning

The CPN model [12] introduced in this paper is focused on the sequence planning aspects of arrival management. The sequence planning phase is the natural starting point for the modeling of an AMANs behavior, as it precedes the other

planning phases (trajectory generation and metering) whose planning is based on the assumed sequence (see Fig. 1). Important decisions about the outcome of the overall process (particularly in terms of punctuality, costs and fuel consumption) are made during the sequence planning stage when the position of an aircraft in the sequence is determined because these decisions can later only be altered under certain circumstances and with considerable effort. Besides these operational aspects, the sequence planning process is also the most straightforward to model and analyze in the form of a Coloured Petri Net since Sequence Planning is inherently a graph theoretic problem with discrete results (a sequence). By contrast the later stages of arrival management (i.e. trajectory generation) contain comparably larger shares of continuous-time computations which may more adequately be described by other modeling approaches.

The task of sequence planning itself can be further divided into the following four subtasks, which form the different parts of the CPN model presented below (Fig. 2):

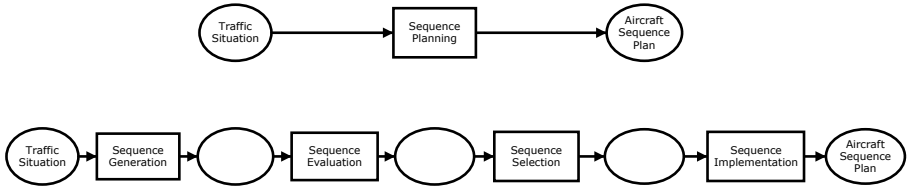


Fig. 2. Four-step scheme of sequence planning

The four subtasks will first be described with regard to the general problem statement of each step before the concrete modelling of the different planning stages in the form of a Coloured Petri Net model is presented in the following section.

2.1 Sequence Generation

The general function of the sequence generation step consists in creating a set of candidate sequences for a number of aircraft within a defined timeframe. Basically this is achieved by calculating all possible permutations of aircraft within a certain timeframe (using a “sliding window” technique) and then computing the respective arrival times for each aircraft of a candidate sequence while maintaining the required temporal and spatial separation between consecutive aircraft. Note however that for set of *NoOfAc* aircraft one will get

$$NoOfCseq = NoOfAc! \quad (1)$$

candidate sequences. This means 720 candidates for a moderate set of 6 aircraft and 40320 candidates for a operationally more relevant set of 8 aircraft.

2.2 Sequence Evaluation

Given the set of candidate sequences the task of the evaluation step is to compute for each candidate a quality value which reflects the desirability/feasibility of practically realizing that candidate sequence from an operational point of view. In order to compute that quality value the evaluation step makes use of a number of evaluation functions which rate among other things (i) if an aircraft in the sequence is able to reach its submitted earliest time of arrival, (ii) if an aircraft is able to land before its submitted latest time of arrival and thus without the need for entering a holding, or (iii) if an aircraft would leave a potential holding pattern according to its relative position in the holding etc.. Another important aspect which is also rated by the evaluation step is if (iv) the “new” candidate sequence is stable with regard to the “old” sequence which has been implemented in the preceding planning cycle. In the ideal case the old and the new sequence are identical, meaning that all aircraft maintain their exact position in the sequence and their target time of arrival (TTA). But if the TTA of an aircraft is changed according to the candidate sequence this would be rated negatively as it would generally mean additional workload for the controller and pilot to rearrange the sequence and might also hamper flight efficiency. The operational requirement for a stability criterion and the associated need to base the selection of a candidate sequence by some means or another on the results of the preceding planning cycle make the whole sequence planning a recursive process.

Mathematically this can be formulated as follows:

$$s_i = f(x_i, s_{i-1}) \quad (2)$$

s_i = Sequence implemented in step i

x_i = Traffic situation and planning input in step i

s_{i-1} = Sequence implemented in step i-1

The recursive nature of the function adds significantly to the complexity of the planning process because it makes the consequences of modifying/adding certain rating functions or adjusting the weighting between these functions harder to predict. The recursion also makes it more difficult to foresee the effects of certain behavior of individual actors (e.g. content of submitted data, estimates provided, time of data submission from airborne side) on the system behavior. The ability to examine these effects in a controlled fashion is one of the main purposes of the CPN model introduced below.

2.3 Sequence Selection

Given the set of candidate sequences together with the ratings computed in the evaluation step, the task of the sequence selection step is to pick the one sequence which will finally be implemented in practice. It is unlikely (mainly for human factors reasons) that the selection task will be totally automated in the near future and that the selection will be executed without the chance for human intervention. This is because the final responsibility for the decision

will for a long time remain with the human controller, thus the controller will also have to be in control. A more realistic scenario might therefore consist in an automated support system pre-selecting a favorable candidate sequence under nominal conditions but allowing the human controller to either intervene and impose constraints on that sequence or even ignore the proposal altogether where necessary.

2.4 Sequence Implementation

The practical implementation of the sequence is to a large degree a manual controller task. The controller gives instructions and clearances to the flightcrew with regard to the aircraft arrival routing, flight profile and speed. The cockpit crew acknowledges the clearances and configures the aircraft to follow the instructed flight path. Nowadays nearly all communication between air traffic control and cockpit crew is still carried out via voice radio. In future this verbal communication might more and more be substituted by a datalink connection.

As mentioned above when discussing the recursive nature of the planning process, as soon as a sequence is implemented it becomes an important input for the next planning cycle executed thereafter.

3 The CPN Model of Cooperative Sequence Planning

This section presents the CPN Tools model [12] developed to analyze the behavior of the cooperative sequence planning process as outlined above. The entire model consists of 16 pages arranged in 5 hierarchical layers. All 16 pages form a tree structure departing from the toplevel page named *SeqPlanning*. An overview of the structure of the model is depicted in Fig. 3.

A special simulation and analysis approach was developed for the model, which also impacts the model structure. To realize that approach, the pages are organized into three different modules ($M1$, $M2$, $M3$) as indicated in Fig. 3. The approach includes alternating and iterative use of state space analysis techniques and simulation for the three different modules. The results of the simulation or analysis phase of one module serve as an initialization for the next phase, realizing a cyclic process (see Fig. 5).

As an outline, Module $M1$ represents variations of aircraft submitted arrival times, the generation of potential candidate sequences and the evaluation of these candidates. A state space based approach is most suitable for this task since it allows that only the constraints with regard to time changes and for sequence building are modeled in $M1$. The computation of the full set of possible combinations is solved through automated state space generation. Module $M2$ performs a selection of candidate sequences generated in $M1$. In order to do that it queries the state space of module $M1$ and compares the candidate sequences with each other. $M2$ is executed in simulation mode. Module $M3$ integrates the results from the iteration between $M1$ and $M2$ and is executed in state space mode.

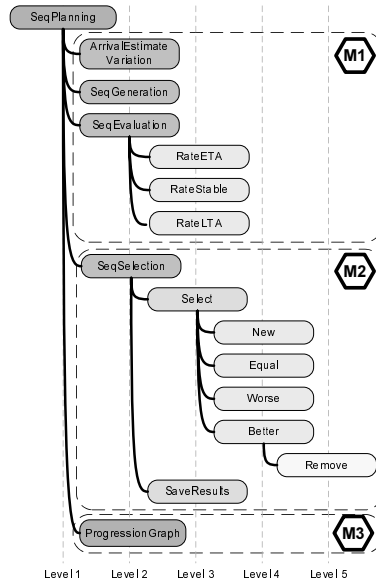


Fig. 3. Page hierarchy of the CPN model

The description starts with the toplevel page *SeqPlanning* in section 3.1, combined with an overview over the special analysis and simulation approach. Sections 3.2 to 3.7 contain detailed descriptions of the subpages of the *SeqPlanning* page.

3.1 *SeqPlanning* Page and Alternating State Space Analysis and Simulation Approach

The *SeqPlanning* page is depicted in Fig. 4. It features five main substitution transitions and two regular transitions.

The substitution transitions *SequenceGeneration*, *SequenceEvaluation*, *SequenceSelection* and the regular transition *SequenceImplementation* together implement the behavior of the automated sequence planning system as introduced in section 2, subsection 2.1 to 2.4 respectively. Regarding the purpose of the four transitions, the model thereby closely corresponds with the four-step scheme of sequence planning shown in Fig. 2. Together the four transitions map a perceived traffic situation as planning input to an implemented sequence plan as an output of the planning system.

The substitution transition *ArrivalEstimateVariation* in the lower part of the picture represents the behavior of aircraft. These aircraft may vary their submitted arrival time estimates at certain points of the arrival procedure. Note that these variations are external to the planning system and represent changes of the traffic situation that the planning system has to react to.

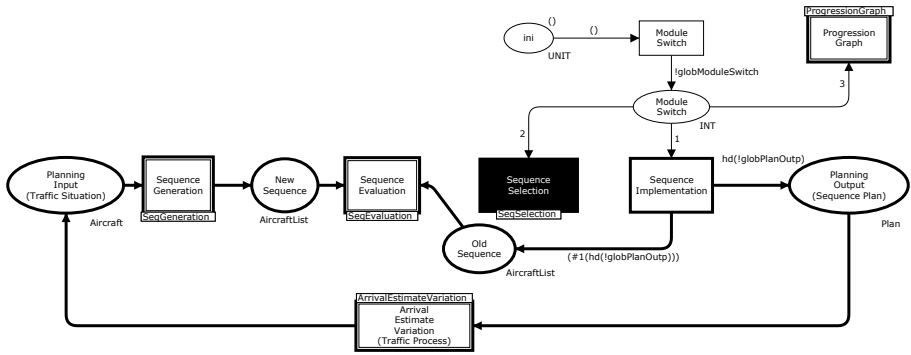


Fig. 4. SeqPlanning Page

While the transitions discussed so far are geometrically arranged as a closed control loop, there appear to be missing links in the net topology of the *SeqPlanning* page, because not all the transitions are actually connected by places and arcs. Concretely, there is no connection between the *SequenceEvaluation* transition and the *SequenceSelection* transition, and also no connection between the *SequenceSelection* transition and the *SequenceImplementation* transition. This would generally be expected in order to close the loop and allow a flow of tokens between these transitions.

The reason for not having the *SequenceSelection* transition connected with arcs as expected is related to the special simulation and analysis approach applied for the exploration of the CPN model. The key points of the approach are here described briefly. For the details on the technical implementation of the developed modelling approach in CPN tools the reader is referred to [13].

A core feature of the approach is the structuring of the overall model into a number of different modules as indicated in Fig. 3. During the execution of the overall model only one module is ‘executable’ at each point in time, meaning that it is only possible for transitions of this executable module to become enabled. The different modules are executed alternately in a controlled order, either in the mode of state space calculation or simulation. This permits to treat specific problems modeled in the different modules in the mode (simulation or state space based) that suits them best and is most efficiently modeled. Information is exchanged between the different modules either via reference variables or by one module making a query on other modules’ previously calculated state space. The resulting control flow for the presented model is illustrated in the activity chart in Fig. 5.

The control flow for the execution of the different modules is defined outside the CPN Model itself through a function in a dedicated SML structure *ClSimSsa*. The structure is loaded into the CPN Tools simulator and the function is then executed via the CPN Tools GUI. To realize the switching between the execution of the different modules and to make a single module executable while deactivating the other modules, the method makes use of a reference variable

Listing 1.1. Non-standard colorsets

```

colset Signature=STRING;
colset Index=INT;
colset SignatureIndex=product Signature*Index;
colset Configuration=STRING;
colset SigConfig=product Signature*Configuration;
colset SigConfigList=list SigConfig;
colset Callsign=STRING;
colset Callsigns= list Callsign;
colset TTA=INT;
colset LTA=INT;
colset ETA=INT;
colset POS=INT;
colset Aircraft=product Callsign*ETA*LTA*TTA*POS;
colset ResPosTTA=STRING;
colset AircraftList= list Aircraft;
colset QUAL=INT;
colset QualVec=product QUAL*QUAL*QUAL;
colset WinLos=product CONFIG*QUAL*QualVec*ResPosTTA*AC_list*Signature;
colset WinLosList=list WinLos;
colset WinLosInt=product WinLos*INT;
colset WinLosIntList=list WinLosInt;
colset Plan=product AC_list*Signature;

```

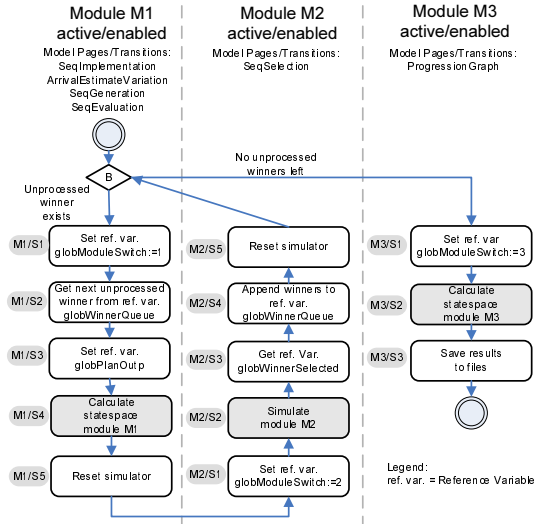


Fig. 5. Activity chart for alternating state space analysis and simulation

globModuleSwitch of type integer. This reference variable is read out on the output arc of the transition *ModuleSwitch* (see *SeqPlanning* page Fig. 4 to determine the marking of the place *ModuleSwitch*. According to the value of the integer token (1, 2 or 3), only one of the three transitions (*SequenceImplementation*, *SequenceSelection* or *ProgressionGraph*) connected to this place can fire and thus only one of the three modules will become executable. On the output arc of the *SequenceImplementation* transition the reference variable *globPlanOutp* is used to initialize the state of the sequence plan for each cycle.

As a convention for the following model description, places in one module (concerns M1) whose marking is queried by other modules are shaded in black. Also transitions in one module (concerns M2) which trigger queries on another module's state space are shaded in black. All non-standard colorset declarations for the model are given in listing 1.1.

3.2 ArrivalEstimateVariation (M1)

The *ArrivalEstimateVariation* page is depicted in Fig. 6. The page models potential changes in the submitted ETAs and LTAs of controlled aircraft. It thereby computes the *input configurations* which describe the potential evolvement of the traffic situation between one iteration and the next. Each input configuration defines a potential traffic situation by specifying the ETAs and LTAs of all involved aircraft at a certain point in time. As part of Module M1, the *ArrivalEstimateVariation* page is executed in state space mode. The page specifies essentially the physical constraints and rules for potential changes of ETA and LTA, the state space computation takes care of computing all possible combinations of their occurrence in the sequence.

The page receives via the input port *PlanningOutput* the output of the last planning cycle. This is a token of colorset *PLAN* consisting of a planned sequence of aircraft with their respective ETAs, LTAs and TTAs and a unique signature (identifier) for this sequence. It returns via the output port on the place *PlanningInput* the input for the next planning cycle. In order to calculate it, two operations are realized.

First, the transition *SplitAircraft*, via the function *filterAircraft* on the output arcs, divides the complete sequence into two subsets of aircraft. The first subset encompasses all aircraft which will not change their submitted ETAs and LTAs during that planning cycle. The tokens representing these aircraft flow directly to the *PlanningInput* (*Fixed Aircraft*) place. The second subset encompasses all aircraft which may change their ETAs and LTAs for the next iteration. These tokens flow to the place *VariableAircraft*. The splitting is defined by the user through the lists of callsigns on places *variableCallsigns* and *fixedCallsigns*.

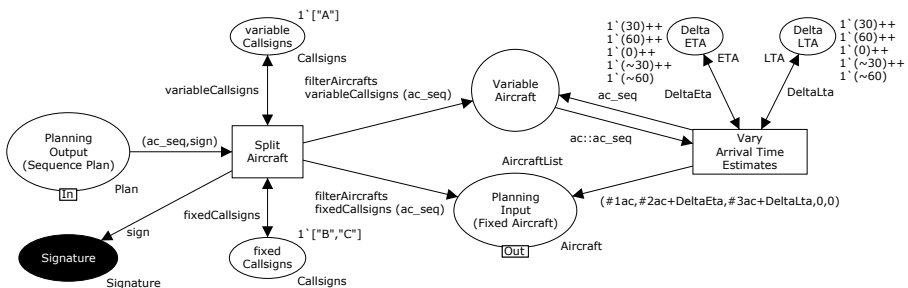


Fig. 6. ArrivalEstimateVariation Page

Second, for the variable aircraft the transition *VaryArrivalTimeEstimates* varies the ETA and LTA by a certain *DeltaEta* and *DeltaLta*. The possible values for *DeltaEta* and *DeltaLta* are defined on the respective places. Aircraft which have received variation of ETA or LTA through the *VaryArrivalEstimate* transition are also deposited on the *PlanningInput(Fixed Aircraft)* place. This leads eventually to a situation where the complete input configuration for all aircraft is fixed on this place.

On the place *Signature* one token of colorset *Signature* is placed which uniquely identifies the parent sequence which the varied input configuration was derived from. This value is queried by a state space query in module *M2* (see section 3.6) and is used in module *M3* to keep track of the planning behavior over various planning cycles.

3.3 SeqGeneration Page (M1)

The SeqGeneration page (Fig. 7) generates (through state space calculation) the full set of candidate sequences by computing all different permutations for a set of aircraft assuming the different input configurations. The number of candidate sequences is

$$NoOfCseq = NoOfInputConfigurations * NoOfAircraft! \quad (3)$$

The page receives a multiset of Aircraft with fixed ETAs and LTAs via the input port place *PlanningInput*. The permutation is realized through the transition *GenerateSequence*. The transition may bind the tokens of the received aircraft multiset in an arbitrary order and assembles them to a fixed sequence on the place *PartialSequence*. A counter realized through place *EarliestTTA* sets the aircrafts target times of arrival (TTA) so that a minimum time interval of 75 time units is maintained between TTAs of two consecutive aircraft. Since the *SeqGeneration* page is part of module *M1* and executed in state space mode, the resulting state space will contain all possible permutations of the sequence. Once the sequence length of the *PartialSequence* equals a constant named *numb_ac* the transition *finished* fires and transfers the *PartialSequence* to the output place *NewSequence*.

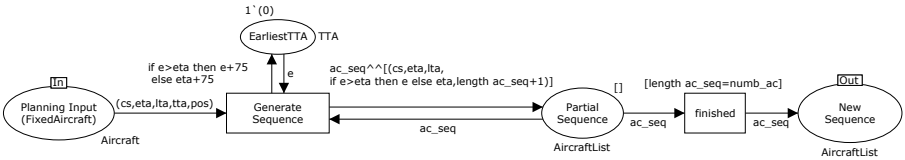


Fig. 7. SeqGeneration page

3.4 SeqEvaluation Page (M1)

The SeqEvaluation page is shown in Fig. 8. The purpose of the page is to assign a quality value to each candidate sequence. In order to do so, its first function is

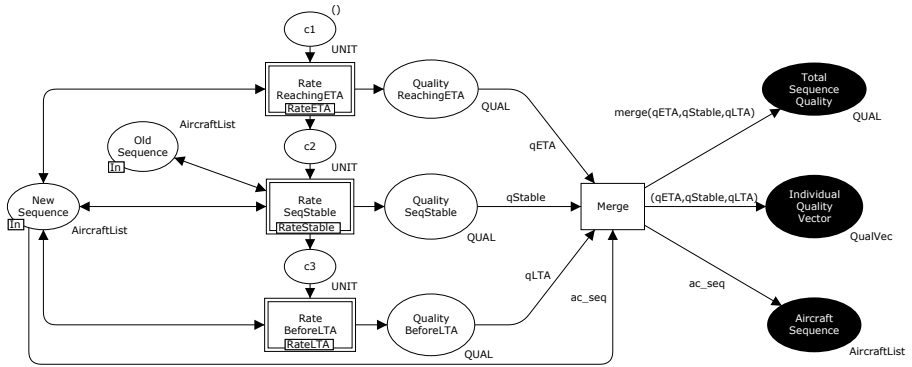


Fig. 8. SeqEvaluation page

to call the different rating functions (substitution transitions *RateReachingETA*, *RateSeqStable*, *RateBeforeLTA*) used to evaluate the candidate sequences. It then merges (transition *Merge*) the individual quality values calculated by those rating functions to one total quality value on place *TotalSequenceQuality*. This value will later decide which candidate sequence is actually selected and implemented.

As the evaluation of stability criterion *RateSeqStable* depends on the sequence implemented in the preceding planning cycle, the *SeqEvaluation* page has two main input places, the *NewSeq* place, which receives the candidate sequence to be evaluated, and the *OldSeq* place with the sequence implemented before. The control flow over places *c1-c3* serves to avoid unnecessary concurrency between the different rating functions.

After the firing of the *Merge* transition there will be no more enabled transition in module *M1*, leading to a dead marking. All dead markings of *M1s* state space will subsequently be investigated by module *M2*. For the places *TotalSequenceQuality*, *IndividualQualityVector* and *AircraftSequence* the marking will be queried to find and select the candidate sequence with the highest quality value.

3.5 Pages RateETA, RateStable, RateLTA (M1)

Pages *RateETA*, *RateStable*, *RateLTA* implement the individual rating functions used to evaluate the candidate sequences. No detailed description of the exact mathematical formulation of the individual functions can be given here for confidentiality reasons. However, in general criterion *RateETA* will rate whether an aircraft in the sequence is able to reach its submitted earliest time of arrival, while *RateLTA* will rate if an aircraft is able to land before its submitted latest time of arrival and thus without the need for entering a holding pattern. The stability criterion responsible for the recursive nature of the sequence planning process is implemented on page *RateStable* (see also section 2.2 sequence evaluation).

3.6 SeqSelection Page (M2)

The *SeqSelection* page (Fig. 9) is a direct subpage of the *SeqPlanning* page and a part of module *M2*. In contrast to *M1*, Module *M2* is executed in simulation mode. Module *M1* (presented above) has previously generated and evaluated all candidate sequences through state space generation. The task of the *SeqSelection* page is to select from all candidate sequences the sequence(s) with the best total quality values for each possible *input configuration* (Fig. 2, step three). These are the sequences which would in practice be implemented by the system (Fig. 2, step four), assuming that the system is not overruled by a differing decision of the air traffic controller.

In Fig. 9 it can be seen that the *SeqSelection* page receives as its only direct input an integer token from the above *SeqPlanning* page via the input port place *ModuleSwitch*. If the value of the token equals 2, it makes the *SeqSelection* page executable by enabling the *init* transition. All other data regarding the set of candidate sequences generated in module *M1* is gained by making queries on module *M1*'s previously generated state space. No further data is received via input arcs.

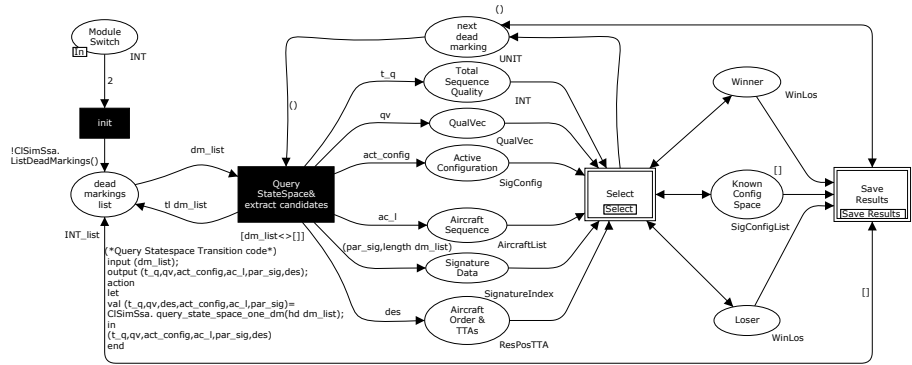


Fig. 9. SeqSelection Page

The output arc of the *init* transition evaluates a function *ListDeadMarkings* defined within the *ClSimSsa* structure. This function returns a list with node numbers of all dead markings of module *M1*'s state space. Each dead marking within the list represents a state where a candidate sequence has been generated and assigned a quality value on the page *SeqEvaluation*. With each firing of the *QueryStateSpace&ExtractCandidates*, the associated code segment subsequently queries one single node of the list and thereby retrieves the marking of the places *TotalSequenceQuality*, *IndividualQualityVector* and *AircraftSequence* page *SeqEvaluation* and place *Signature* (page *ArrivalEstimateVariation*). For the node, the resulting information characterizing the candidate sequence and its quality is deposited as tokens on the output places of the *QueryStateSpace&ExtractCandidates*.

The job of the *Select* transition is to find among all candidate sequences and for each input configuration the “winner(s)”, that is the sequence(s) with the highest quality value. Thus, it runs one by one through all the candidates while always storing the best known candidate(s) for each input configuration on the *Winner* place and dumping the tokens representing candidates with lower quality values on the *Loser* Place. In detail, this comparison and data handling is realized by the nets on the subpages *new*, *equal*, *worse*, *better*, and *remove* of *Select* transition, which will not be further explained here.

Once all dead markings (or candidate sequences respectively) of the state space have been inspected and the optimal sequence for each input configuration has been found, the transition and its associated subpage *SaveResults* writes the results of the analysis to different text files. Three different write-in-file monitors are thereby used to save the winning candidates, the losing candidates and a description of the examined input configurations in three separate files. Some preliminary results of such an analysis and possible ways of visualizing the results are discussed in section 4.1.

3.7 ProgressionGraph Page (M3)

The *ProgressionGraph* page implements module *M3*. It is executed in state space mode. In addition to the write-in-file monitors of the *SaveResults* page, the *ProgressionGraph* page (Fig. 10) provides a second means to access and process the results of the simulation. While the monitor-based solution of the *SaveResults* page considers the behavior of one single sequence planning cycle in isolation, the *ProgressionGraph* serves to integrate the results of a number of consecutive planning cycles. In order to do that, the *ProgressionGraph* relies on a string signature which is saved with every sequence (every “winning sequence” in particular). This signature makes it possible to identify from which parent sequence (implemented in the preceding planning cycle) this particular winning sequence derived.

The transitions on the *ProgressionGraph* page implement three different actions:

- The page is enabled via the single input port *ModuleSwitch*. The transition *init* reads out the reference variable *globWinnerQueue*, which represents a list of all winning sequences recorded over time during the cyclic execution of modules *M1* and *M2*. The *init* transition deposits the very first element in the list (the parent sequence of all other sequences) on the place *Predecessor*. All other sequences are deposited as a multiset of sequences on the place *Successor*.
- The transition *progress to Successor* can bind and interchange two tokens from places *predecessor* and *successor* under the condition (guard) that, according to their signatures, the predecessor sequence represents the parent sequence of the successor sequence. This traces the possible evolvement of the sequence over time.
- The transition *interchange with equally configured sequence* can bind and interchange two tokens if they describe sequences with identical input con-

figurations (identical ETAs and LTAs as input to the planning system) but with different results for the order of the sequence (different response of the planning system). This detects situations with different planning system responses to the same input data due to the time-dependent behavior and recursivity of the process.

A result for a state space created with the ProgressionGraph page visualized with a modified version of the OGtoGraphViz interface of CPN Tools is presented and discussed in section 4.2.

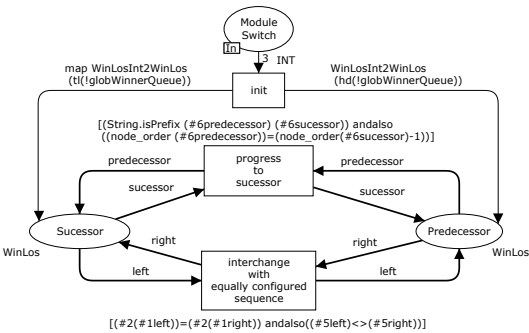


Fig. 10. ProgressionGraph page

4 Preliminary Simulation and Analysis Results

The section presents some preliminary results achieved through simulations and analysis of the above model. The results contain both output produced with the monitor-based analysis to examine one single planning cycle as well as with the second approach realized on the *ProgressionGraph* page to examine the behavior of the system over a number of consecutive planning cycles.

For the purpose of this paper a very small example sequence has been chosen, which contains only three aircraft (“A”, “B”, “C”). It is assumed that the initial state of the sequence planning process is as shown in Table 1.

Table 1. Initial state of example sequence

Aircraft	ETA ²	LTA ³	TTA ⁴	POS ⁵
A	110	240	110	1
B	120	250	185	2
C	123	207	260	3

² ETA=Submitted Earliest Time of Arrival, that is earliest time the aircraft could reach the airport according to its own estimation.
³ LTA=Submitted Latest Time of Arrival, that is latest time the aircraft could reach the airport according to its own estimation (without entering a holding).
⁴ TTA=Target Time of Arrival planned by sequence planner.
⁵ POS=Position of aircraft in the sequence planned by sequence planner.

All time values for ETA, LTA and TTA are assumed to be in seconds, only the relative differences between those numbers are practically relevant, however, the absolute values are arbitrary.

4.1 Monitor-Based Analysis of a Single Planning Cycle

For the monitor-based analysis it is assumed that aircraft “A” and “B” are free to vary their ETAs before the next planning cycle for one of the following values: $\Delta\text{Eta} \in \{-90, -60, -30, 0, 30, 60, 90\}$. Meanwhile the ETA of “C” as well as the LTAs of all three aircraft “A”, “B”, “C” remains constant. Choosing a negative ΔEta (earlier ETA) in practice represents a situation in which either an aircraft experiences tailwinds or in which it chooses a slightly higher speed signifying higher fuel consumption but a potentially shorter flight time (assuming there is no delay at the airport). In contrast, choosing a positive ΔEta (later ETA) could mean that an aircraft experiences headwinds or chooses a more economic cruising speed paying the price of a potentially delayed arrival time.

The purpose of the simulation is then to examine how the revised ETA influences the aircrafts’ position in the sequence as planned by the AMAN, as well as the target time of arrival the sequence planner calculates for each aircraft on the basis of the new input configuration.

Fig. 11 shows the planned target time of arrival of aircraft “A”, “B” and “C” plotted against the submitted earliest time of arrival of “A” and “B”. The TTAs actually simulated with the CPN models are represented by the triangles, squares and circles for “A”, “B” and “C” respectively. The mesh surface was fitted to the measured data afterwards in order to grasp their position in three-dimensional space more easily. It has to be noted with regard to the interpretation of the

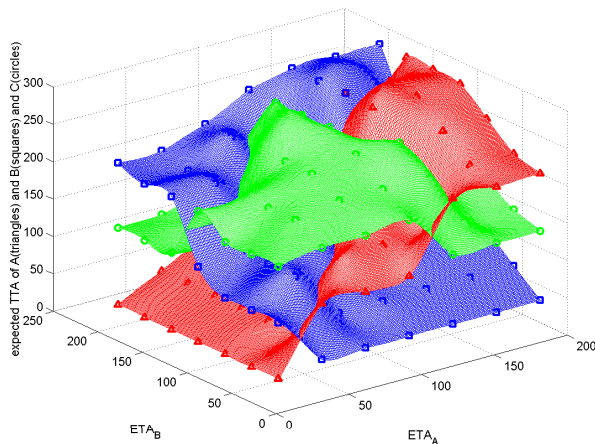


Fig. 11. Planned target time of arrival (TTA) as a function of submitted earliest time of arrival (ETA) of “A” and “B” (ETA of “C” constant)

interpolated surfaces in Fig. 11 that the intersections between surfaces are not valid in a strict sense (since that would mean more than one aircraft with the same target time of arrival, which would necessarily violate separation criteria) and are due to the characteristics of the interpolation function. The order of the surface layers in Fig. 11 also allows to read off the order of aircraft in the sequence. From the perspective of the individual aircraft the gradient of the surface can be interpreted as an incentive to adjust its behavior (speed/estimates) in one way or the other [14,15]. From the perspective of the air traffic control (or the perspective of the AMAN alternatively) the most upper surface of the three is generally the most interesting, as it reveals the time when the last aircraft in the considered sequence will be landed.

4.2 Analysis of System Behavior over Consecutive Planning Cycles

Fig. 12 shows a fragment of a graph describing the system behavior of the sequence planner over two consecutive planning cycles in reaction to variations of the ETAs and LTAs. The same initial state of the example sequence is assumed as in section 4.1. The range of potential values for *DeltaEta* was further constrained to be $\Delta\eta \in \{-60, 0, +60\}$ in the first planning cycle and $\Delta\eta \in \{0, +60\}$ in the second planning cycle. As in section 4.1 it is assumed that aircraft “A” and “B” are free to vary their earliest time of arrival while aircraft “C” is obliged to keep its ETA constant. The latest time of arrival (LTAs) of all three aircraft is also held constant in both planning cycles.

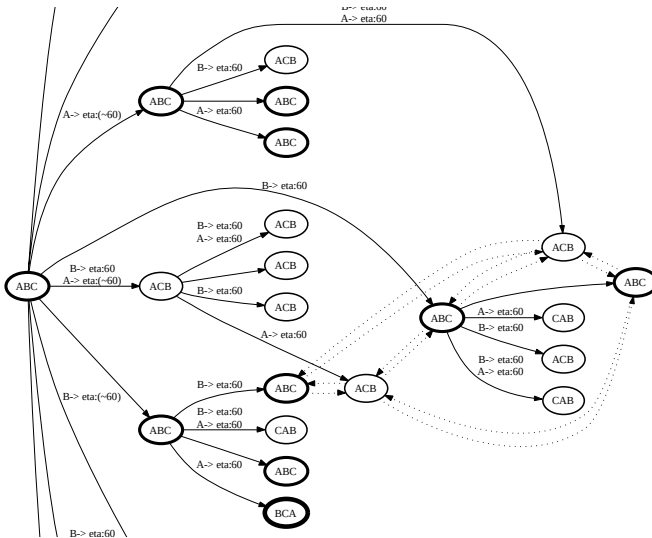


Fig. 12. Planned sequence as a function of submitted earliest time of arrival over two consecutive planning cycles

The *ProgressionGraph* is generated through state space calculation of module M3. The visualization of the Graph is realized using the GraphViz software [16] with a modified version of the OGtoGraphViz interface supplied with CPN Tools.

The graph (Fig. 12) depicts in its nodes the description of the sequences itself, (thus indicating the position of each aircraft in the sequence). The edge labels contain the value *DeltaEta* by which the ETA is varied between one planning step and the next. The linewidth of arcs is used to code branches and nodes of a tree which are favorable from the perspective of a certain aircraft. In Fig. 12, the more favourable the position for aircraft “B”, the broader the line surrounding the label of the node is drawn. A second type of edge drawn in dotted lines is used to connect nodes which are “equally configured”, that is which contain sequences with identical input configuration, but with a different order of aircraft implemented by the sequence planner. The edges between equally configured nodes are caused by the transition *SameConfig* on the *ProgressionGraph* page. Where such edges exist between two nodes, they naturally run in both directions. In Fig. 12 it can be seen that dotted edges exist between the first-order node where aircraft “B” has revised the ETA by $\Delta\text{Eta}=+60$ and second-order node where “A” in the first planning cycle has revised its ETA by $\Delta\text{Eta}=-60$ and then in the second planning cycle both “A” and “B” have revised their ETA by $\Delta\text{Eta}=+60$. The addition of the revision values over the two consecutive cycles in both cases leads to the same configuration (thus the two nodes are “equally configured”), the resulting sequence, however, is “ABC” for the first-order node and “ACB” for the second order node. Thus by switching “A”’s ETA back and forth, and then ending up with the same configuration, we have come to result different from that achieved by keeping “A”’s ETA constant.

5 Conclusions

The paper introduces a CPN model of a potential future sequence planning process for cooperative arrival planning in air traffic control. The basis of the cooperation consists in that the airborne side, that is the involved aircraft, provide information for example with regard to their estimated earliest and latest times of arrival at the airport. This information is then used by the sequence planner (on the ground) to establish a favourable sequence in which the aircraft will be led to the runway. The model was built in order to achieve a better understanding of how the sequence planning responds to certain behavior of individual actors/aircraft involved in the process. The basic sequence planning procedure assumed and modeled in this paper (i.e. the core sequence planning algorithm) was developed at DLR and has also been implemented in a prototype system using conventional programming languages [17,2]. It can be argued that the model duplicates some functionality of the existing prototype. The duplication is justified by the effective and intuitive means of analyzing the model provided with CPN Tools. Some of the benefits with respect to analysis have been pointed out in section 4. For example nodes have been detected in the state space which represented identical input configurations of all aircraft, but have led to differing

responses of the planning system due to the time-dependent nature of the planning process. The extensive possibilities and flexible manner of analysing the system behavior of the CP-net appear as a strong argument in favor of the CPN model. To be able to more easily analyze the planning system's behavior over a number of consecutive planning cycles a novel simulation and analysis method has been developed for this work. The method allows the automated and repetitive (re)entering of the CPN-Tools state space tool and implementation of a closed loop process between state space analysis and simulation. Through this method it will in future be possible to analyze and discuss more complex effects on longer sequences of aircraft and over an extensive number of planning cycles.

References

1. Fairclough, I.: Phare Advanced Tools Arrival Manager Final Report. Technical report, Eurocontrol, European Organisation for Safety of Air Navigation (1999)
2. Zielinsky, T.: Erkennung und Lösung von Konflikten bei Anflug-Trajektorien. Technical report, German Aerospace Center (DLR) (2003)
3. Büchner, U., Czerlitzki, B., Hansen, H., Helmke, H., Pahner, S., Pfeil, A., Schnell, M., Schnieder, H., Theis, P., Uebbing-Rumke, M.: KOPIM-AIRCRAFT - Entwicklungstatus von boden- und bordseitigen Systemen und von operationellen ATM-Verfahren und Konzepten für ein kooperatives ATM. Technical report, German Aerospace Center (DLR), Airbus Deutschland GmbH, TU Darmstadt (2005)
4. Korn, B., Helmke, H., Kuenz, A.: 4D Trajectory Management in the Extended TMA: Coupling AMAN and 4D FMS for Optimized Approach Trajectories. In: ICAS 2006 - 25th International Congress of the Aeronautical Sciences, Hamburg, Germany (2006)
5. Butler, R., Carreno, V., Di Vito, B., Hayhurst, K., Holloway, C., Miner, P., Munoz, C., Geser, A., Gottliebsen, H.: NASA Langley's Research and Technology-Transfer Program in Formal Methods. Technical report, NASA Langley (2002)
6. Butler, R., Geser, A., Maddalon, J., Munoz, C.: Formal Analysis of Air Traffic Management Systems: The Case of Conflict Resolution and Recovery. In: Winter Simulation Conference (WSC 2003), New Orleans (2003)
7. Munoz, C., Dowek, G., Carreno, V.: Modelling and Verification of an Air Traffic Concept of Operations. In: International Symposium on Software Testing and Analysis, Boston, Massachusetts, pp. 175–182 (2004)
8. Degani, A., Heymann, M.: Formal Verification of Human-Automation Interaction. *Human Factors* 44(1), 28–43 (2002)
9. Carreno, V., Munoz, C.: Formal Analysis of Parallel Landing Scenarios. In: Digital Avionics System Conferences, Philadelphia, USA, vol. 1, pp. 175–182 (2000)
10. Werther, B., Moehlenbrink, C., Rudolph, M.: Coloured Petri Net based Formal Airport Control Model for Simulation and Analysis of Airport Control Processes. In: Duffy, V.G. (ed.) HCII 2007 and DHM 2007. LNCS, vol. 4561. Springer, Heidelberg (2007)
11. Kovacs, A., Nemeth, E., Hangos, K.: Modeling and Optimization of Runway Traffic Flow Using Coloured Petri Nets. In: International Conference on Control and Automation (ICCA), Budapest, Hungary, vol. 2, pp. 881–886 (2005)
12. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *Software Tools for Technology Transfer (STTT)* 9(3-4), 213–254 (2007)

13. Oberheid, H., Gamrad, D., Söffker, D.: Closed Loop State Space Analysis and Simulation for Cognitive Systems. In: 8th International Conference on Application of Concurrency to System Design (submitted, 2008)
14. Günther, T., Fricke, H.: Potential of Speed Control on Flight Efficiency. In: ICRAT - Second International Conference on Research in Air Transportation, pp. 197–201 (2006)
15. Günther, T.: Validierung der Interdependenzen eines Systems zur Ankunftszeitoptimierung von Flugzeugen in Ergänzung zu einem Arrival Management an einem Verkehrsflughafen. Diploma thesis, Technical University Dresden (2004)
16. AT&T-Research: GraphViz Manual (2006),
<http://www.graphviz.org/Documentation.php>
17. Schwarz, D.: Anflugsequenzplanung mit dem A* Algorithmus zur Beschleunigung der Sequenzsuche. Technical Report 112-2005/02, German Aerospace Center (DLR) (2005)

Process Discovery Using Integer Linear Programming

J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik

Department of Mathematics and Computer Science

Technische Universiteit Eindhoven

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

{j.m.e.m.v.d.werf,b.f.v.d.dongen,c.a.j.hurkens,a.serebrenik}@tue.nl

Abstract. The research domain of *process discovery* aims at constructing a process model (e.g. a Petri net) which is an abstract representation of an execution log. Such a Petri net should (1) be able to reproduce the log under consideration and (2) be independent of the number of cases in the log. In this paper, we present a process discovery algorithm where we use concepts taken from the language-based theory of regions, a well-known Petri net research area. We identify a number of shortcomings of this theory from the process discovery perspective, and we provide solutions based on integer linear programming.

1 Introduction

Enterprise information systems typically log information on the steps performed by the users of the system. For legacy information systems, such execution logs are often the only means for gaining insight into ongoing processes. Especially, since system documentation is usually missing or obsolete and nobody is confident enough to provide such documentation. Hence, in this paper we consider the problem of *process discovery* [7], i.e. we construct a process model describing the processes controlled by the information system by simply using the execution log. We restrict our attention to the control flow, i.e., we focus on the ordering of activities executed, rather than on the data recorded.

Table 1 illustrates our notion of an event log, where it is important to realize that we assume that every event recorded is related to a single execution of a process, also referred to as a *case*.

A process model (in our case a Petri net) discovered from a given execution log should satisfy a number of requirements. First of all, such a Petri net should be capable of reproducing the log, i.e. every sequence of events recorded in the log should correspond to a firing sequence of the Petri net. Second, the size of the Petri net should be independent of the number of cases in the log. Finally, the Petri net should be such that the places in the net are as expressive as possible in terms of the dependencies between transitions they express.

A problem similar to process discovery arises in areas such as hardware design and control of manufacturing systems. There, the so called *theory of regions* is used to construct a Petri net from a behavioral specification (e.g., a language),

Table 1. An event log

case id	activity id	originator	case id	activity id	originator
case 1	activity A	John	case 5	activity A	Sue
case 2	activity A	John	case 4	activity C	Carol
case 3	activity A	Sue	case 1	activity D	Pete
case 3	activity B	Carol	case 3	activity C	Sue
case 1	activity B	Mike	case 3	activity D	Pete
case 1	activity C	John	case 4	activity B	Sue
case 2	activity C	Mike	case 5	activity E	Claire
case 4	activity A	Sue	case 5	activity D	Claire
case 2	activity B	John	case 4	activity D	Pete
case 2	activity D	Pete			

such that the behavior of this net corresponds with the specified behavior (if such a net exists).

In this paper we investigate the application of the theory of regions in the field of process discovery. It should be noted that we are not interested in Petri nets whose behavior corresponds completely with the given execution log, i.e. logs cannot be assumed to exhibit all behavior possible. Instead, they merely provide insights into “common practice” within a company.

In Section 3, we show that a straightforward application of the theory of regions to process discovery would lead to Petri nets of which the number of places depends on size of the log. Therefore, in Section 4, we discuss how ideas from the theory of regions can be combined with generally accepted concepts from the field of process discovery to generate Petri nets that satisfy the requirements given above. Furthermore, using the log of Table 1 as an illustrative example, we show how our approach can lead to Petri nets having certain structural properties, such as marked graphs, state machines and free-choice nets [14]. Finally, in Section 5 we present the implementation of our approach in ProM [4], followed by a brief discussion on its usability on logs taken from practice. In Section 6, we provide some conclusions.

2 Preliminaries

Let S be a set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' | S' \subseteq S\}$. A *bag* (*multiset*) m over S is a function $S \rightarrow \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \dots\}$ denotes the set of natural numbers. The set of all bags over S is denoted by $\mathbb{N}^S$. We identify a bag with all elements occurring only once with the set containing these elements, and vice versa. We use $+$ and $-$ for the sum and difference of two bags, and $=, <, >, \leq, \geq$ for the comparison of two bags, which are defined in a standard way. We use $\emptyset$ for the empty bag, and $\in$ for the element inclusion. We write e.g. $m = 2[p] + [q]$ for a bag m with $m(p) = 2$, $m(q) = 1$ and $m(x) = 0$, for all $x \notin \{p, q\}$. We use the standard notation $|m|$ and $|S|$ to denote the number of elements in bags and sets. Let $n \in \mathbb{N}$. A *sequence* over S of length n is a

function $\sigma: \{1, \dots, n\} \rightarrow S$. If $n > 0$ and $\sigma(1) = a_1, \dots, \sigma(n) = a_n$, we write $\sigma = \langle a_1, \dots, a_n \rangle$, and σ_i for $\sigma(i)$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the empty sequence, and is denoted by ϵ . The set of finite sequences over S is denoted by S^* . Let $v, \tau \in S^*$ be two sequences. Concatenation, denoted by $\sigma = v \cdot \tau$ is defined as $\sigma: \{1, \dots, |v| + |\tau|\} \rightarrow S$, such that for $1 \leq i \leq |v|$, $\sigma(i) = v(i)$, and for $|v| + 1 \leq i \leq |\sigma|$, $\sigma(i) = \tau(i - |v|)$. Further, we define the prefix $\leq$ on sequences by $v \leq \tau$ if and only if there exists a sequence $\rho \in S^*$ such that $\tau = v \cdot \rho$. We use $\mathbf{x}$ to denote column vectors and for a sequence $\sigma \in S^*$, the *Parikh vector* $\boldsymbol{\sigma}: S \rightarrow \mathbb{N}$ defines the number of occurrences of each element of S in the sequence, i.e. $\boldsymbol{\sigma}(s) = |\{i | 1 \leq i \leq |\sigma|, \sigma(i) = s\}|$, for all $s \in S$. A subset $\mathcal{L} \subseteq S^*$ is called a *language* over S . It is *prefix-closed* if and only if for all sequences $\sigma = \sigma' \cdot a \in \mathcal{L}$ holds $\sigma' \in \mathcal{L}$, for any $a \in S$.

Definition 2.1. (Petri net) A *Petri net* N is a 3-tuple $N = (P, T, F)$, where (1) P and T are two disjoint sets of *places* and *transitions* respectively; we call the elements of the set $P \cup T$ *nodes* of N ; (2) $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*; an element of F is called an *arc*.

Let $N = (P, T, F)$ be a Petri net. Given a node $n \in P \cup T$, we define its preset $\bullet_N n = \{n' | (n', n) \in F\}$, and its postset $n_N \bullet = \{n' | (n, n') \in F\}$. If the context is clear, we omit the N in the subscript.

To describe the execution semantics of a net, we use *markings*. A marking m of a net $N = (P, T, F)$ is a bag over P . Markings are states (configurations) of a net. A pair (N, m) is called a *marked Petri net*. A transition $t \in T$ is *enabled* in a marking $m \in \mathbb{N}^P$, denoted by $(N, m)[t]$ if and only if $\bullet t \leq m$. Enabled transitions may *fire*. A transition firing results in a new marking m' with $m' = m - \bullet t + t \bullet$, denoted by $(N, m)[t] (N, m')$.

Definition 2.2. (Firing sequence) Let $N = (P, T, F)$ be a Petri net, and (N, m) be a marked Petri net. A sequence $\sigma \in T^*$ is called a *firing sequence* of (N, m) if and only if for $n = |\sigma|$, there exist markings $m_1, \dots, m_{n-1} \in \mathbb{N}^P$ and transitions $t_1, \dots, t_n \in T$ such that $\sigma = \langle t_1, \dots, t_n \rangle$, and, $(N, m)[t_1] (N, m_1) \dots (N, m_{n-1})[t_n]$. We lift the notations for being enabled and firing to firing sequences, i.e. if $\sigma \in T^*$ be a firing sequence, then for all $1 \leq k \leq |\sigma|$ and for all places $p \in P$ it holds that $m(p) + \sum_{i=1}^{k-1} (\sigma_i \bullet(p) - \bullet \sigma_i(p)) \geq \bullet \sigma_k(p)$.

As we stated before, a single execution of a model is called a *case*. If the model is a Petri net, then a single firing sequence that results in a dead marking (a marking where no transition is enabled) is a case. Since most information systems log all kinds of events during execution of a process, we establish a link between an execution log of an information system and firing sequences of Petri nets. The basic assumption is that the log contains information about specific *transitions* executed for specific *cases*.

Definition 2.3. (Case, Log) Let T be a set of transitions, $\sigma \in T^*$ is a *case*, and $L \in \mathcal{P}(T^*)$ is an *execution log* if and only if for all $t \in T$ holds that there is a $\sigma \in L$, such that $t \in \sigma$. In other words, an execution log is a set of firing sequences of some marked Petri net $(N, m) = ((P, T, F), m)$, where P , F and m are unknown and where no transition is dead.

In Definition 2.3, we define a log as a *set* of cases. Note that in real life, logs are *bags* of cases, i.e. a case with a specific order in which transitions are executed may occur more than once, as shown in our example. However, in this paper, we do not have to consider occurrence frequencies of cases and therefore sets suffice.

Recall that the goal of process discovery is to obtain a Petri net that can reproduce the execution log under consideration. In [11], Lemma 1 states the conditions under which this is the case. Here, we repeat that Lemma and we adapt it to our own notation.

Definition 2.4. (Replayable log) Let $L \in \mathcal{P}(T^*)$ be an execution log, and $\sigma \in L$ a case. Furthermore, let $N = ((P, T, F), m)$ be a marked Petri net. If the log L is a subset of all possible cases of (N, m) , i.e. each case in L is a firing sequence in (N, m) $((N, m)[\sigma])$, we say that L can be replayed by (N, m) .

In order to construct a Petri net that can indeed reproduce a given log, the theory of regions can be used. In Section 3, we present this theory in detail and we argue why the classical algorithms in existence are not directly applicable in the context of process discovery. In Section 4, we show how to extend the theory of regions to be more applicable in a process discovery context.

3 Theory of Regions

The general question answered by the theory of regions is: given the specified behavior of a system, what is the Petri net that represents this behavior? Both the form in which the behavior is specified as well as the “represents” statement can be expressed in different ways. Mainly, we distinguish two types, the first of which is state-based region theory [9,12,16]. This theory focusses on the synthesis of Petri nets from state-based models, where the statespace of the Petri net is branching bisimilar to the given state-based model. Although state-based region theory can be applied in the process discovery context [6], the main problem is that execution logs rarely carry state information and the construction of this state information from a log is far from trivial [6].

In this paper, we consider language-based region theory [8,13,17], of which [17] presents a nice overview. In [17], the authors show how for different classes of languages (step languages, regular languages and partial languages) a Petri net can be derived such that the resulting net is the smallest Petri net in which the words in the language are possible firing sequences.

In this section, we introduce the language-based regions, we briefly show how the authors of [11,17] used these regions in the context of process discovery and why we feel that this application is not a suitable one.

3.1 Language-Based Theory of Regions

Given a prefix-closed language $\mathcal{L}$ over some non-empty, finite set T , the language-based theory of regions tries to find a finite Petri net $N(\mathcal{L})$ in which the transitions correspond to the elements in the set T and of which all sequences in

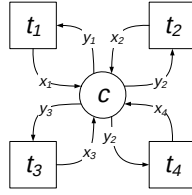


Fig. 1. Region for a log with four events

the language are firing sequences. Furthermore, the Petri net should have only minimal firing sequences not in the language.

The Petri net $N(\mathcal{L}) = (\emptyset, T, \emptyset)$ is a finite Petri net in which all sequences are firing sequences. However, its behavior not minimal. Therefore, the behavior of this Petri net needs to be reduced, such that the Petri net still allows to reproduce all sequences in the language, but does not allow for more behavior. This is achieved by adding places to the Petri net. The theory of regions provides a method to calculate these places, using *regions*.

Definition 3.1. (Region) A region of a prefix-closed language $\mathcal{L}$ over T is a triple $(\mathbf{x}, \mathbf{y}, c)$ with $\mathbf{x}, \mathbf{y} \in \{0, 1\}^T$ and $c \in \{0, 1\}$, such that for each non-empty sequence $w = w' \cdot a \in \mathcal{L}$, $w' \in \mathcal{L}$, $a \in T$:

$$c + \sum_{t \in T} (\mathbf{w}'(t) \cdot \mathbf{x}(t) - \mathbf{w}(t) \cdot \mathbf{y}(t)) \geq 0$$

This can be rewritten into the inequation system:

$$c \cdot \mathbf{1} + A' \cdot \mathbf{x} - A \cdot \mathbf{y} \geq \mathbf{0}$$

where A and A' are two $|\mathcal{L}| \times |T|$ matrices with $A(w, t) = \mathbf{w}(t)$, and $A'(w, t) = \mathbf{w}'(t)$, with $w = w' \cdot a$. The set of all regions of a language is denoted by $\mathfrak{R}(\mathcal{L})$ and the region $(\mathbf{0}, \mathbf{0}, 0)$ is called the *trivial region*.¹

Figure 1 shows a region for a log with four events, i.e. each solution $(\mathbf{x}, \mathbf{y}, c)$ of the inequation system can be regarded in the context of a Petri net, where the region corresponds to a feasible place with preset $\{t | t \in T, \mathbf{x}(t) = 1\}$ and postset $\{t | t \in T, \mathbf{y}(t) = 1\}$, and initially marked with c tokens. Note that we do not assume arc-weights, where the authors of [8, 11, 13, 17] do. However, in process modelling languages, such arc weights typically do not exist, hence we decided to ignore them. Our approach can however easily be extended to incorporate them.

Since the place represented by a region is a place which can be added to a Petri net, without disturbing the fact that the net can reproduce the language under consideration, such a place is called a *feasible place*.

¹ To reduce calculation time, the inequation system can be rewritten to the form $[\mathbf{1}; A'; -A] \cdot \mathbf{r} \geq \mathbf{0}$ which can be simplified by eliminating duplicate rows.

Definition 3.2. (Feasible place) Let $\mathcal{L}$ be a prefix-closed language over T and let $N = ((P, T, F), m)$ be a marked Petri net. A place $p \in P$ is called *feasible* if and only if there exists a *corresponding* region $(\mathbf{x}, \mathbf{y}, c) \in \mathfrak{R}(\mathcal{L})$ such that $m(p) = c$, and $\mathbf{x}(t) = 1$ if and only if $t \in \bullet p$, and $\mathbf{y}(t) = 1$ if and only if $t \in p^\bullet$.

In [11, 17] it was shown that any solution of the inequation system of Definition 3.1 can be added to a Petri net without influencing the ability of that Petri net to replay the log. However, since there are infinitely many solutions of that inequation system, there are infinite many feasible places and the authors of [11, 17] present two ways of finitely representing these places.

Basis representation. In the basis representation, the set of places is chosen such that it is a basis for the non-negative integer solution space of the linear inequation system. Although such a basis always exists for homogeneous inequation systems, it is worst-case exponential in the number of equations [17]. By construction of the inequation system, the number of equations is linear in the number of traces, and thus, the basis representation is worst-case exponential in the number of events in the log. Hence, an event log containing ten thousand events, referring to 40 transitions, might result in a Petri net containing a hundred million places connected to those 40 transitions. Although [11] provides some ideas on how to remove redundant places from the basis, these procedures still require the basis to be constructed fully. Furthermore, the implementation of the work in [11] is not publicly available for testing and no analysis is presented of this approach on realistically sized logs.

Separating representation. To reduce the theoretical size of the resulting Petri net, the authors of [11, 17] propose a separating representation. In this representation, places that separate the allowed behavior (specified by the system) using words not in the language are added to the resulting Petri net. Although this representation is no longer exponential in the size of the language, but polynomial, it requires the user to specify undesired behavior, which can hardly be expected in the setting of process discovery, i.e. nothing is known about the behavior, except the behavior seen in the event log. Furthermore, again no analysis is presented of this approach on realistically sized logs.

In [11, 17] the authors propose the separating representation for regular and step languages and by this, they maximize the number of places generated by the number of events in the log times the number of transitions. As we stated in the introduction, the size of the Petri net should not depend on the size of the log for the approach to be applicable in the context of process discovery.

4 Integer Linear Programming Formulation

In [17, 11] it was shown that any solution of the inequation system of Definition 3.1 can be added to a Petri net without influencing the ability of that Petri net to replay the log. Both the basis and the separating representation are presented to select which places to indeed add to the Petri net. However, as shown

in Section 3, we argue that the theoretical upper bound on the number of places selected is high. Therefore, we take a different selection mechanism for adding places that:

- Explicitly express certain causal dependencies between transitions that can be discovered from the log, and
- Favors places which are more expressive than others (i.e. the added places restrict the behavior as much as possible).

In this section, we first show how to express a log as a prefix-closed language, which is a trivial, but necessary step and we quantify the expressiveness of places, in order to provide a target function, necessary to translate the inequation system of Definition 3.1 into a integer linear programming problem in Subsection 4.2. In Section 4.3, we show a first algorithm to generate a Petri net. In Subsection 4.4, we then provide insights into the causal dependencies found in a log and how these can be used for finding places. We conclude this section with a description of different algorithms for different classes of Petri nets.

4.1 Log to Language

To apply the language-based theory of regions in the field of process discovery, we need to represent the process log as a prefix-closed language, i.e. by all the traces present in the process log, and their prefixes. Recall from Definition 2.3 that a process log is a finite set of traces.

Definition 4.1. (Language of a process log) Let T be a set of activities, $L \in \mathcal{P}(T^*)$ a process log over this set of transitions. The language $\mathcal{L}$ that represents this process log, uses alphabet T , and is defined by:

$$\mathcal{L} = \{l \in T^* \mid \exists l' \in L : l \leq l'\}$$

As mentioned before, a trivial Petri net capable of reproducing a language is a net with only transitions. To restrict the behavior allowed by the Petri net, but not observed in the log, we start adding places to that Petri net. However, the places that we add to the Petri net should be *as expressive as possible*, which can be expressed using the following observation. If we remove the arc (p, t) from F in a Petri net $N = (P, T, F)$ (assuming $p \in P, t \in T, (p, t) \in F$), the resulting net still can replay the log (as we only weakened the pre-condition of transition t). Also if we would add a non-existing arc (t, p) to F , with $t \in T$ and $p \in P$, the resulting net still can replay the log as it strengthens the post-condition of t .

Lemma 4.2. (Adding an incoming arc to a place retains behavior) Let $N = ((P, T, F), m)$ be a marked Petri net that can replay the process log $L \in \mathcal{P}(T^*)$. Let $p \in P$ and $t \in T$ such that $(t, p) \notin F$. The marked Petri net $N' = ((P, T, F'), m)$ with $F' = F \cup \{(t, p)\}$ can replay the log L .

Proof. Let $\sigma = \sigma_1 \cdot t \cdot \sigma_2 \in T^*$, such that $t \notin \sigma_1$ be a firing sequence of (N, m) . Let $m' \in \mathbb{N}^P$ such that $(N, m) [\sigma_1 \cdot t] (N, m')$ and $(N, m') [\sigma_2]$. We know that for all $p' \in t_N^\bullet$ holds that $m'(p') > 0$, since t just fired. Assume $t \notin \sigma_1$. Then

$(N', m)[\sigma_1 \cdot t](N', m'')$ with $m'' = m' + [p]$. Furthermore, since $m' + [p] > m'$, we know that $(N', m'')[\sigma_2]$. By induction on the occurrences of t , we get $(N', m)[\sigma]$. $\square$

Lemma 4.3. (Removing an outgoing arc from a place retains behavior)

Let $N = ((P, T, F), m)$ be a marked Petri net that can replay the process log $L \in \mathcal{P}(T^*)$. Let $p \in P$ and $t \in T$ such that $(p, t) \in F$. The marked Petri net $N' = ((P, T, F'), m)$ with $F' = F \setminus \{(p, t)\}$ can replay the log L .

Proof. Let $\sigma = \sigma_1 \cdot t \cdot \sigma_2 \in T^*$, such that $t \notin \sigma_1$ be a firing sequence of (N, m) . Let $m' \in \mathbb{N}^P$ such that $(N, m)[\sigma_1](N, m')$. We know that for all $p' \in_N^\bullet t$ holds that $m'(p') > 0$, since t is enabled. This implies that for all $p' \in_{N'}^\bullet t$ also holds that $m'(p') > 0$, since $\bullet_{N'} t = \bullet_N t \setminus \{p\}$. Hence, $(N', m)[\sigma_1](N', m')$ and $(N', m')[t](N', m'')$. Due to monotonicity, we have $(N', m'')[\sigma_2]$. Hence $(N', m)[\sigma]$. $\square$

Besides searching for regions that lead to places with maximum expressiveness, i.e. a place with a maximum number of input arcs and a minimal number of output arcs, we are also searching for “minimal regions”. As in the region theory for synthesizing Petri nets with arc weights, minimal regions are regions that are not the sum of two other regions.

Definition 4.4. (Minimal region) Let $L \in \mathcal{P}(T^*)$ be a log, let $r = (\mathbf{x}, \mathbf{y}, c)$ be a region. We say that r is *minimal*, if there do not exist two other, non trivial, regions r_1, r_2 with $r_1 = (\mathbf{x}_1, \mathbf{y}_1, c_1)$ and $r_2 = (\mathbf{x}_2, \mathbf{y}_2, c_2)$, such that $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$ and $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2$ and $c = c_1 + c_2$.

Using the inequation system of Definition 3.1, the expressiveness of a place and the partial order on regions, we can define an integer linear programming problem (ILP formulation [20]) to construct the places of a Petri net in a logical order.

4.2 ILP Formulation

In order to transform the linear inequation system introduced in Section 3 to an ILP problem, we need a target function. Since we have shown that places are most expressive if their input is minimal and their output is maximal, we could minimize the value of a target function $f((\mathbf{x}, \mathbf{y}, c)) = c + \mathbf{1}^\top \cdot \mathbf{x} - \mathbf{1}^\top \cdot \mathbf{y}$, i.e. adding an input arc from the corresponding feasible place increases the value of $\mathbf{1}^\top \cdot \mathbf{x}$, as does removing an output arc from the corresponding feasible place.

However, it is easy to show that this function does not favor minimal regions, i.e. regions that are no sum of two other regions. Therefore, we need a target function f' , such that for any three regions r_1, r_2 and r_3 with $r_3 = r_1 + r_2$ holds that $f'(r_1) < f'(r_3)$ and $f'(r_2) < f'(r_3)$, while preserving the property that regions corresponding to more expressive feasible places have a lower target value. We propose the function $c + \mathbf{1}^\top(\mathbf{1} \cdot c + A \cdot (\mathbf{x} - \mathbf{y}))$, which is derived as follows. From Definition 3.1, we have $c + \sum_{t \in T} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)) \geq 0$ for all non-empty sequences $w \in \mathcal{L}$, and since $c \geq 0$, this also holds for the empty sequence

$\epsilon \in \mathcal{L}$. Therefore, $\sum_{w \in \mathcal{L}} (c + \sum_{t \in T} (\mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)))) \geq 0^2$. Rewriting gives: $\sum_{w \in \mathcal{L}} (c + \sum_{t \in T} (\mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)))) = c + \sum_{w \in (\mathcal{L} \setminus \{\epsilon\})} (c + \sum_{t \in T} (\mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t))))$, which we can reformulate to $c + \mathbf{1}^\top (\mathbf{1} \cdot c + A \cdot (\mathbf{x} - \mathbf{y}))$.

Definition 4.5. (Target function) Let $L \in \mathcal{P}(T^*)$ be a log, let $r = (\mathbf{x}, \mathbf{y}, c)$ be a region and let A be the matrix as in definition 3.1. We define a target function $\tau : \mathfrak{R}(\mathcal{L}) \rightarrow \mathbb{N}$, such that $\tau(r) = c + \mathbf{1}^\top (\mathbf{1} \cdot c + A \cdot (\mathbf{x} - \mathbf{y}))$.

We show that the target function of Definition 4.5 indeed satisfies all criteria.

Lemma 4.6. (Target function favors minimal regions) Let $L \in \mathcal{P}(T^*)$ be a log, let $r_1 = (\mathbf{x}_1, \mathbf{y}_1, c_1)$, $r_2 = (\mathbf{x}_2, \mathbf{y}_2, c_2)$ and $r_3 = (\mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2, c_1 + c_2)$ be three non-trivial regions (i.e. with $r_3 = r_1 + r_2$). Furthermore, let A be the matrix defined in Definition 3.1. Let $\tau : \mathfrak{R}(\mathcal{L}) \rightarrow \mathbb{N}$, such that $\tau(r) = c + \mathbf{1}^\top (\mathbf{1} \cdot c + A \cdot (\mathbf{x} - \mathbf{y}))$. Then $\tau(r_1) < \tau(r_3)$ and $\tau(r_2) < \tau(r_3)$.

Proof. We proof this lemma by showing that $\tau(r) > 0$ holds for any non-trivial region $r \in \mathfrak{R}(\mathcal{L})$.

Let $r = (\mathbf{x}, \mathbf{y}, c) \in \mathfrak{R}(\mathcal{L})$ be a non-trivial region. By Definition 3.1, we have $\tau(r) \geq 0$, since $A(\sigma, t) \geq A'(\sigma, t)$ for any $\sigma \in \mathcal{L}$ and $t \in T$. If $\mathbf{x} = \mathbf{y} = \mathbf{0}$, then $c = 1$, and hence $\tau(r) > 0$. If there is a transition $t_0 \in T$ such that $\mathbf{x}(t_0) > 0$, then there are sequences $w, w' \in \mathcal{L}$ such that $w = w' \cdot t_0$, and hence $\mathbf{w}(t_0) > \mathbf{w}'(t_0)$. Since $\sum_{t \in T} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)) = \mathbf{w}(t_0)(\mathbf{x}(t_0) - \mathbf{y}(t_0)) + \sum_{t \in T, t \neq t_0} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t))$ and $\mathbf{w}(t_0)(\mathbf{x}(t_0) - \mathbf{y}(t_0)) > \mathbf{w}'(t_0)\mathbf{x}(t_0) - \mathbf{w}(t_0)\mathbf{y}(t_0)$, we know $\sum_{w \in \mathcal{L}} \sum_{t \in T} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)) > \sum_{w \in \mathcal{L}} \sum_{t \in T} (\mathbf{w}'(t)\mathbf{x}(t) - \mathbf{w}(t)\mathbf{y}(t))$. Therefore, $\sum_{w \in \mathcal{L}} (c + \sum_{t \in T} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t))) > \sum_{w \in \mathcal{L}} (c + \sum_{t \in T} (\mathbf{w}'(t)\mathbf{x}(t) - \mathbf{w}(t)\mathbf{y}(t))) \geq 0$. If $\mathbf{x} = \mathbf{0}$, then there is a transition $t_0 \in T$ such that $\mathbf{y}(t_0) > 0$, and a sequence $w \in \mathcal{L}$ such that $w(t_0) > 0$. Furthermore, $c + \sum_{t \in T} \mathbf{w}(t)(\mathbf{x}(t) - \mathbf{y}(t)) = c - \sum_{t \in T} \mathbf{w}(t)\mathbf{y}(t)$. Then $c - \sum_{t \in T} \mathbf{w}(t)\mathbf{y}(t) = c - \mathbf{w}(t_0) - \sum_{t \in T, t \neq t_0} \mathbf{w}(t)\mathbf{y}(t)$. Since $\mathbf{w}(t_0) \geq 1$, and $\sum_{t \in T, t \neq t_0} \mathbf{w}(t)\mathbf{y}(t) \geq 0$, we have $c > 0$. Since $\mathbf{1}^\top (\mathbf{1} \cdot c + A \cdot (\mathbf{x} - \mathbf{y})) \geq 0$, we have by definition of τ , $\tau(r) \geq c > 0$.

Hence $\tau(r) > 0$ for any non trivial $r \in \mathfrak{R}(\mathcal{L})$. $\square$

Lemma 4.6 shows that this target function satisfies our requirements for non-trivial regions. This does not restrict the generality of our approach, since places without arcs attached to them are of no interest for the behavior of a Petri net.

Combining Definition 3.1 with the condition set by Lemma 4.6 and the target function of Definition 4.5, we get the following integer linear programming problem.

Definition 4.7. (ILP formulation) Let $L \in \mathcal{P}(T^*)$ be a log, and let A and A' be the matrices defined in Definition 3.1. We define the ILP ILP_L corresponding with this log as:

² We cannot rewrite this to the matrix notation as in Definition 3.1, since matrix A uses only non-empty sequences, while the empty sequence is in $\mathcal{L}$.

Minimize $c + \mathbf{1}^\top \cdot A \cdot (\mathbf{x} - \mathbf{y})$	Defintion 4.5
such that $c + A' \cdot \mathbf{x} - A \cdot \mathbf{y} \geq \mathbf{0}$	Definition 3.1
$\mathbf{1}^\top \cdot \mathbf{x} + \mathbf{1}^\top \cdot \mathbf{y} \geq 1$	There should be at least one edge
$\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$	$x \in \{0, 1\}^{ T }$
$\mathbf{0} \leq \mathbf{y} \leq \mathbf{1}$	$y \in \{0, 1\}^{ T }$
$0 \leq c \leq 1$	$c \in \{0, 1\}$

The ILP problem presented in Definition 4.7 provides the basis for our process discovery problem. However, an optimal solution to this ILP only provides a single feasible place with a minimal value for the target function. Therefore, in the next subsection, we show how this ILP problem can be used as a basis for constructing a Petri net from a log.

4.3 Constructing Petri Nets Using ILP

In the previous subsection, we provided the basis for constructing a Petri net from a log. In fact, the target function of Definition 4.5 provides a partial order on all elements of the set $\mathfrak{R}(\mathcal{L})$, i.e. the set of all regions of a language. In this subsection, we show how to generate the first n places of a Petri net, that is (1) able to reproduce a log under consideration and (2) of which the places are as expressive as possible.

A trivial approach would be to add each found solution as a negative example to the ILP problem, i.e. explicitly forbidding this solution. However, it is clear that once a region r has been found and the corresponding feasible place is added to the Petri net, we are no longer interested in regions r' for which the corresponding feasible place has less tokens, less outgoing arcs or more incoming arcs, i.e. we are only interested in unrelated regions.

Definition 4.8. (Refining the ILP after each solution) Let $L \in \mathcal{P}(T^*)$ be a log, let A and A' be the matrices defined in Definition 3.1 and let $ILP_{(L,0)}$ be the corresponding ILP. Furthermore, let region $r_0 = (\mathbf{x}_0, \mathbf{y}_0, c_0)$ be a minimal solution of $ILP_{(L,0)}$. We define the refined ILP as $ILP_{(L,1)}$, with the extra constraint specifying that:

$$-c_0 \cdot c + \mathbf{y}^\top \cdot (\mathbf{1} - \mathbf{y}_0) - \mathbf{x}^\top \cdot \mathbf{x}_0 \geq -c_0 + 1 - \mathbf{1}^\top \cdot \mathbf{x}_0$$

Lemma 4.9. (Refining yields unrelated regions) Let $L \in \mathcal{P}(T^*)$ be a log, let A and A' be the matrices defined in Definition 3.1 and let $ILP_{(L,0)}$ be the corresponding ILP. Furthermore, let region $r_0 = (\mathbf{x}_0, \mathbf{y}_0, c_0)$ be a minimal solution of $ILP_{(L,0)}$ and let $r_1 = (\mathbf{x}_1, \mathbf{y}_1, c_1)$ be a minimal solution of $ILP_{(L,1)}$, where $ILP_{(L,1)}$ is the refinement of $ILP_{(L,0)}$ following Definition 4.8. Then, $c_1 < c_0$ or there exists a $t \in T$, such that $x_1(t) < x_0(t) \vee y_1(t) > y_0(t)$.

Proof. Assume that $c_1 \geq c_0$ and for all $t \in T$ holds that $x_1(t) \geq x_0(t)$, and $y_1(t) \leq y_0(t)$. Then since $c_0, c_1 \in \{0, 1\}$ we know that $c_0 \cdot c_1 = c_0$. Similarly, $\mathbf{y}_1^\top \cdot (\mathbf{1} - \mathbf{y}_0) = 0$ and $\mathbf{x}_0^\top \cdot \mathbf{x}_1 = \mathbf{x}_0^\top \cdot \mathbf{1}$. Hence $-c_0 \cdot c_1 + \mathbf{y}_1^\top \cdot (\mathbf{1} - \mathbf{y}_0) - \mathbf{x}_0^\top \cdot \mathbf{x}_1 = -c_0 - \mathbf{x}_0^\top \cdot \mathbf{1}$ and since $-c_0 - \mathbf{x}_0^\top \cdot \mathbf{1} < -c_0 + 1 - \mathbf{1}^\top \cdot \mathbf{x}_0$, we know that r_1 is not a solution of $ILP_{(L,0)}$. This is a contradiction. $\square$

The refinement operator presented above, basically defines an algorithm for constructing the places of a Petri net that is capable of reproducing a given log. The places are generated in an order which ensures that the most expressive places are found first and that only places are added that have less tokens, less outgoing arcs, or more incoming arcs. Furthermore, it is easy to see that the solutions of each refined ILP are also solutions of the original ILP, hence all places constructed using this procedure are feasible places.

The procedure, however, still has the downside that the total number of places introduced is worst-case exponential in the number of transitions. Furthermore, the first n places might be introduced linking a small number of transitions, whereas other transitions in the net are only linked after the first n places are found. Since there is no way to provide insights into the value of n for a given Petri net, we propose a more suitable approach, *not using the refinement step of Definition 4.8*. Instead, we propose to guide the search for solutions (i.e. for places) by metrics from the field of process discovery [5, 7, 15, 21].

4.4 Using Log-Based Properties

Recall from the beginning of this section, that we are specifically interested in places expressing explicit causal dependencies between transitions. In this subsection, we first introduce how these causal dependencies are usually derived from a log file. Then we use these relations in combination with the ILP of Definition 4.7 to construct a Petri net.

Definition 4.10. (Causal dependency [7]) Let T be a set of transitions and $L \in \mathcal{P}(T^*)$ an execution log. If for two activities $a, b \in T$, there are traces $\sigma_1, \sigma_2 \in T^*$ such that $\sigma_1 \cdot a \cdot b \cdot \sigma_2 \in L$, we write $a >_L b$. If in a log L we have $a >_L b$ and not $b >_L a$, there is a *causal dependency* between a and b , denoted by $a \rightarrow_L b$.

In [7], it was shown that *if* a log L satisfies a certain completeness criterion and *if* there exists a Petri net of a certain class [7] that can reproduce this log, then the $>_L$ relation is enough to reconstruct this Petri net from the log. However, the completeness criterion assumes knowledge of the Petri net used to generate the log and hence it is undecidable whether an arbitrary log is complete or not. Nonetheless, we provide the formal definition of the notion of completeness, and we prove that for complete logs, causal dependencies directly relate to places and hence provide a good guide for finding these places.

Definition 4.11. (Complete log [7]) Let $N = ((P, T, F), m)$ be a marked Petri net. Let $L \in \mathcal{P}(T^*)$ be a process log. The log L is called *complete* if and only if there are traces $\sigma_1, \sigma_2 \in T^*$ such that $(N, m)[\sigma_1 \cdot a \cdot b \cdot \sigma_2]$ implies $a >_L b$.

In [7], the proof that a causal dependency corresponds to a place is only given for safe Petri nets (where each place will never contain more than one token during execution). This can however be generalized for non-safe Petri nets.

Lemma 4.12. (Causality implies a place) Let $N = ((P, T, F), m)$ be a marked Petri net. Let L be a complete log of N . For all $a, b \in T$, it holds that if $a \neq b$ and $a \rightarrow_L b$ then $a^\bullet \cap b^\bullet \neq \emptyset$.

Proof. Assume $a \rightarrow_L b$ and $a^\bullet \cap b^\bullet = \emptyset$. By the definition of $\rightarrow_L$, there exist sequences $\sigma_1, \sigma_2 \in T^*$ such that $(N, m)[\sigma_1 \cdot a \cdot b \cdot \sigma_2]$. Let $s = m + N\sigma_1$, then $(N, s)[a]$, but also $(N, s)[b]$ (N, s') , for some $s' \in \mathbb{N}^P$, since $a^\bullet \cap b^\bullet = \emptyset$. Further we have $\neg(N, s')[a]$, since otherwise $a \not\rightarrow_L b$. Therefore, $(b^\bullet \setminus a^\bullet) \cap a^\bullet \neq \emptyset$. Let $p \in (b^\bullet \setminus a^\bullet) \cap a^\bullet$. Then, $s(p) = 1$, since if $s(p) > 1$, a would be enabled in (N, s') . Therefore, b is not enabled after firing $(\sigma \cdot a)$. This is a contradiction, since now $\neg(N, s)[a \cdot b]$. $\square$

Causal dependencies between transitions are used by many process discovery algorithms [5, 7, 15, 21] and generally provide a good indication as to which transitions should be connected through places. Furthermore, extensive techniques are available to derive causal dependencies between transitions using heuristic approaches [7, 15]. In order to find a place expressing a specific causal dependency, we extend the ILP presented in Definition 4.7.

Definition 4.13. (ILP for causal dependency) Let $L \in \mathcal{P}(T^*)$ be a log, let A and A' be the matrices defined in Definition 3.1 and let ILP_L be the corresponding ILP. Furthermore, let $t_1, t_2 \in T$ and assume $t_1 \rightarrow_L t_2$. We define the refined ILP, $ILP_{(L, t_1 \rightarrow t_2)}$ as ILP_L , with two extra bounds specifying that:

$$x(t_1) = y(t_2) = 1$$

A solution of the optimization problem expresses the causal dependency $t_1 \rightarrow_L t_2$, and restricts the behavior as much as possible. However, such a solution does not have to exist, i.e. the ILP might be infeasible, in which case no place is added to the Petri net being constructed. Nonetheless, by considering a separate ILP for each causal dependency in the log, a Petri net can be constructed, in which each place is as expressive as possible and expresses at least one dependency derived from the log. With this approach at most one place is generated for each dependency and thus the upper bound of places in $N(\mathcal{L})$ is the number of causal dependencies, which is worst-case quadratic in the number of transitions and hence *independent* of the size of the log.

4.5 Net Types

So far, we presented two algorithms for constructing a Petri net able to replay a log, using an ILP formulation. The two algorithms presented are generic and can easily be extended to different log types or to different net classes. In this subsection, we present possible extensions. For all of these extensions, we briefly sketch how they affect the ILP, and what the result is in terms of computational complexity.

Workflow nets. Workflow nets [3] are a special class of Petri nets with a single marked input place and a single output place which are often used for modelling

business processes. To search for workflow nets using our ILP approach, we do not allow for places to be marked, unless they have no incoming arcs. In terms of the ILP, this simply translates into saying that $c = 0$ when searching a place for a causal dependency and to separately search for initial places for each transition t not expressing a causal dependency, but with $c = 1$ and $\mathbf{1}^T \cdot \mathbf{x} = 0$.

Figure 2 shows a Petri net constructed from the log of Table 1. This Petri net is *almost* what we consider to be a workflow net, i.e. it has a clear initial marking and the initially marked place does not have incoming arcs. When replaying the log of Table 1 in this net, it is obvious that the net is empty (i.e. no places contain tokens) after completion of each case, whereas workflow nets should contain a clearly marked final place when a case is complete. This property can also be expressed in terms of constraints, by demanding that the Petri net should have an empty marking after the completion of a case (in most cases, it is then rather easy to extend such a net to a net with a single output place).

Empty after case completion. Another property which is desirable in process discovery, is the ability to identify the final marking. Using the ILP formulation we presented in this paper, this can easily be achieved, by adding constraints

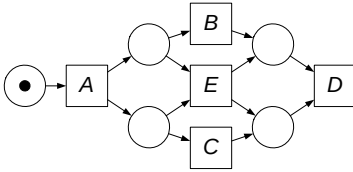


Fig. 2. Workflow net (without output place)

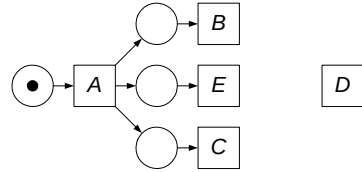


Fig. 3. Marked graph

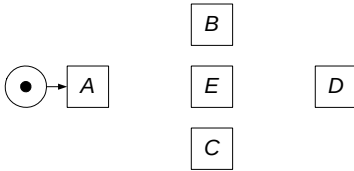


Fig. 4. Marked graph, empty after case completion

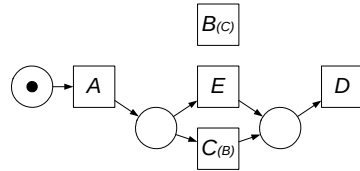


Fig. 5. State machine

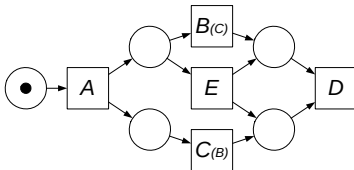


Fig. 6. Free choice

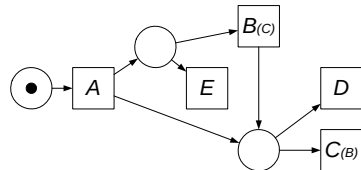


Fig. 7. Free choice, empty after case completion

ensuring that for all cases in the log which are no prefix of any other case in the log (or maximum cases), the net is empty when all transitions are fired. These constraints have the form $c + \sigma^\top \cdot x - a^\top \cdot y = 0$, where σ is the Parikh vector corresponding to a maximum case σ .

The requirement that the net has to be empty after case completion sometimes leads to a reduction of the number of feasible places. Consider for example a class of Petri nets called “marked graphs”.

Marked Graphs. In a marked graph [19], places have at most one input place and one output place, i.e. $\mathbf{1}^\top \cdot x \leq 1$ and $\mathbf{1}^\top \cdot y \leq 1$. The influence of these constraints on the computation time is again negligible, however, these constraints do introduce a problem, since it is no longer possible to maximize the number of output arcs of a place (as it is at most 1). However, the procedure will find as many places with single input single output as possible.

In Figure 3, a marked graph is presented that was constructed from the log of Table 1. This net is clearly capable of replaying the log. However, after completion of a case, tokens remain in the net, either in the place before “E” or in the places before “B” and “C”. When looking for a marked graph that is empty after case completion, the result is the net of Figure 4.

State machines. State machines [19] are the counterpart of Marked graphs, i.e. transitions have at most one input place and one output place. It is easy to see that this cannot be captured by an extension of the ILP directly. The property is not dealing with a single solution of the ILP (i.e. a single place), but it is dealing with the collection of all places found.

Nonetheless, our implementation in ProM [4], which we present in Section 5 does contain a naive algorithm for generating state machines. The algorithm implemented in ProM proceeds as follows. First, the ILP is constructed and solved, thus yielding a place p to be added to the Petri net. Then, this place is added and from that point on, for all transitions $t \in \bullet p$, we say that $x(t) = 0$ and for all transitions $t \in p^\bullet$, we say that $y(t) = 0$. Currently, the order in which places are found is undeterministic: the first place satisfying the conditions is chosen, and from that moment on no other places are connected to the transitions in its pre- and post-set.

Figure 5 shows a possible state machine that can be constructed using the log of Table 1. Note that transitions “B” and “C” are symmetrical, i.e. the figure actually shows 2 possible state machines, of which one will be provided by the implementation.

Free-Choice nets. Similar to state machines, free choice nets [14] impose restrictions on the net as a whole, rather than on a single place. A Petri net is called free choice, if for all of its transitions t_1, t_2 holds that if there exists a place $p \in \bullet t_1 \cap \bullet t_2$ then $\bullet t_1 = \bullet t_2$.

Our implementation allows for the construction of free-choice nets and the algorithm used works as follows. First, all causal dependencies are placed on a stack. Then, for the first dependency on the stack, the ILP is solved, thus

yielding a region $(\mathbf{x}_0, \mathbf{y}_0, c_0)$ with $\mathbf{1}^\top \cdot \mathbf{y}_0 > 1$ corresponding to a place p with multiple outgoing arcs to be added to the Petri net. Then, this place is added and from that point on, constraints are added, saying for all $t_1, t_2 \in T$ with $\mathbf{y}_0(t_1) = \mathbf{y}_0(t_2) = 1$ holds that $\mathbf{y}(t_1) = \mathbf{y}(t_2)$, i.e. all outgoing edges of the place added to the Petri net appear together, or none of them appears. If after this a place p_1 is found with even more outgoing edges than p , then p_1 is added to the Petri net, p is removed, the constraints are updated and the causal dependencies expressed by p , but not by p_1 are placed back on the stack. This procedure is repeated until the stack is empty.

It is easy to see that the algorithm presented above indeed terminates, i.e. places added to a Petri net that call for the removal of existing places always have more outgoing arcs than the removed places. Since the number of outgoing arcs is limited by the number of transitions, there is an upper bound to the number of removals and hence to the number of constraints placed back on the stack. The algorithm does however put a strain on the computation time, since each causal dependency is investigated as most as often as the number of transitions in the log, and hence instead of solving the ILP $|T|^2$ times, it might be solved $|T|^3$ times. However, since the added constraints have a specific form, solving the ILP gets quicker with each iteration (due to reduced complexity of the Branch-and-Bound part) [20].

Figure 6 shows a possible free-choice net that can be constructed using the log of Table 1. Note that transitions “B” and “C” are again symmetrical. Furthermore, the only difference between this net and the net of Figure 2 is that there is no arc from the place between “A” and “C” to “E”. Adding this arc would violate the free-choice property. The fact that this arc is not there however does violate the property that the net is empty after case completion. Figure 7 shows a free-choice net that is empty after case completion. However, this net is no longer a so-called *elementary net*.

Pure nets. Before introducing elementary nets, we first define *pure* nets [18], since elementary nets are pure. In a pure net, no self-loops occur. By adding a constraint $\mathbf{x}(t) + \mathbf{y}(t) \leq 1$, for each transition $t \in T$, each transition either consumes or produces tokens in a place, but not both at the same time. This procedure slightly increases the size of the ILP problem (with as many constraints as transitions found in the log), thus resulting in a slight increase in computation time. However, since most logs have far more prefixes than actual transitions, the overall effect is negligible.

Elementary nets. Elementary Petri nets [17] are nets in which transitions can only fire when their output places are empty. This can easily be worked into the ILP, as shown in [17]. Two sets of constraints are required. First, self-loops are explicitly forbidden since elementary nets are pure and then, by adding the constraints $c + \mathbf{1}^\top \cdot A \cdot \mathbf{x} - \mathbf{1}^\top \cdot A \cdot \mathbf{y} \leq 1$ it is ensured that after firing a transition each of its output places should contain at most one token. State machines, marked graphs and free-choice nets can all be made elementary this way. When requiring an elementary net however, the problem size doubles (there are twice

as many constraints) and since the execution time is exponential in the size of the problem, the worst-case execution time is squared.

In this section, we presented a way of constructing a Petri net from an execution log using an ILP formulation. We presented a large number of net types and extensions to get a Petri net satisfying criteria set by a user. The figures on page 380 nicely show that with different sets of constraints, different models can be produced.

Although we used a toy example to illustrate the different concepts, we introduce our implementation embedded in the process discovery framework ProM, which is capable of constructing nets for logs with thousands of cases referring to dozens of transitions.

5 Implementation in ProM

The (Pro)cess (M)ining framework *ProM* [4] has been developed as a completely pluggable environment for process discovery and related topics. It can be extended by simply adding plug-ins, and currently, more than 200 plug-ins have been added. The ProM framework can be downloaded from www.processmining.org.

In the context of this paper, the “Parikh language-based region miner” was developed, that implements the algorithms presented in Section 4. The Petri nets on Page 380 were all constructed using our plugin. For solving ILP problems, the open-source solver LpSolve [1] is used. Although experiments with CPLEX [2] have been conducted, we feel that the use of the open-source alternative is essential: it allows for distribution of ProM, as well as reproducibility of the results presented in this paper.

Note that the plugin is also capable of dealing with partially ordered logs, i.e. logs where all cases are represented by partial orders on its events. The construction of the ILP in that case is largely the same as for the totally ordered cases as presented in [17, 11] and as included in VIPtool [10].

5.1 Numerical Analysis

Using the implementation in ProM, we performed some performance analysis on our approach. We tested our algorithm on a collection of logs with varying numbers of transitions and varying numbers of cases, using the default settings of our plugin, which include the constraints for elementary nets and empty nets after case completion. Furthermore, we used logs that are distributed with the release of ProM and causal dependencies are used for guiding the solver.

Note that solving an ILP problem consists of two stages. The first stage uses the Simplex algorithm, which is worst-case exponential, but generally outperforms polynomial algorithms, and the second phase, which is an exponential search.

The results presented in Table 2 show that adding cases to a log referring to a given number of transitions increases the necessary calculation time in a

Table 2. Numerical analysis results

log	# transitions	# variables	# cases	# events	# constraints	time (hh:mm:ss.sss)
a12f0n00_1	12	25	200	618	54	0.406
a12f0n00_2			600	1848	54	0.922
a12f0n00_3			1000	3077	54	1.120
a12f0n00_4			1400	4333	54	1.201
a12f0n00_5			1800	5573	54	1.234
a22f0n00_1	22	45	100	1833	1894	1:40.063
a22f0n00_2			300	5698	4350	5:07.344
a22f0n00_3			500	9463	5863	7:50.875
a22f0n00_4			700	13215	7238	10:24.219
a22f0n00_5			900	16952	8405	12:29.313
a32f0n00_1	32	65	100	2549	3352	32:14.047
a32f0n00_2			300	7657	7779	1:06:24.735
a32f0n00_3			500	12717	10927	1:46:34.469
a32f0n00_4			700	17977	13680	2:43:40.641
a32f0n00_5			900	23195	15978	2:54:01.765

seemingly sub-linear fashion³, however this might not be the case in general. Figure 8 shows a screenshot of ProM, showing the Petri nets discovered for the log “a22” with 300 cases in the bottom right corner. The settings dialog for our plugin is shown in the top-left corner.

Since for the log “a12” the number of constraints remains constant for different log sizes, the increase in processing time is only due to the overhead of calculating the causal dependencies, which is polynomial in the size of the log. For the other logs however, the increase in processing time is due to the fact that adding constraints influences the first phase of solving an ILP problem, but not the second phase which is far more complex. Furthermore, our experiments so-far show an almost linear dependency between the number of cases in the log and the execution time, although this might not be the case in general.

On the other hand, when increasing the number of transitions (i.e. the number of variables) instead of the number of cases, the branch-and-bound phase is heavily influenced. Hence the computation time increases exponentially, since the branch-and-bound algorithm is worst-case exponential. However, these results show that the execution time is still feasible, i.e. results for a large log can be generated overnight.

6 Conclusion and Future Work

In this paper we presented a new method for process discovery using integer linear programming (ILP). The main idea is that places restrict the possible firing sequences of a Petri net. Hence, we search for as many places as possible, such that the resulting Petri net is consistent with the log, i.e. such that the Petri net is able to replay the log.

The well-known theory of regions solves a similar problem, but for finite languages. Finite languages can be considered as prefix closures of a log and the

³ These calculations were performed on a 3 GHz Pentium 4, using ProM 4.2 and LpSolve 5.5.0.10 running Java 1.5. The memory consumption never exceeded 256MB.

theory of regions tries to synthesize a Petri net which can reproduce the language as precisely as possible. In [11,17] this idea is elaborated and the problem is formalized using a linear inequation system. However, the Petri nets synthesized using the approach of [11,17] scale in the number of events in the log, which is undesirable in process discovery.

In this paper, we build on the methods proposed in [11,17]. First of all we have defined an optimality criterion transforming the inequation system into an ILP. This ILP is then solved under different conditions, such that the places of a Petri net capable of replaying the log are constructed.

The optimality criterion we defined is such that it guarantees that more expressive places are found first, i.e. places with less input arcs and more output arcs are favored. Furthermore, using the causality relation that is used in the alpha algorithm [7], we are able to specifically target the search to places expressing this relation. This causality relation is generally accepted in the field of process discovery and, under the assumption the log is complete, is shown to directly relate to places. Furthermore, the size of the constructed Petri net is shown to be independent on the number of events in the log, which makes this approach applicable in more practical situations.

Using additional constraints, we can enforce structural net properties of the discovered Petri net, such as the freedom of choice. It is clear that not all these constraints for structural properties lead to feasible solutions, but nonetheless, we always find a Petri net that is consistent with the log. For all our constraints we provide lemmas motivating these constraints.

The numerical quality of our approach is promising: we can discover nets with about 25 transitions and a log with about 1000 cases in about 15 minutes on a standard desktop PC. Moreover, while the execution time of our method appears to scale sub-linear in the size of the log, although this needs to be validated more thoroughly.

Since each place can be discovered in isolation, it seems to be easy to parallelize the algorithm and to use grid computing techniques to speed up the computation time. Other open questions concern the expression of more structural and behavioral properties into linear constraints and the use of this algorithms in a post-processing phase of other algorithms.

References

1. LP Solve Reference guide, <http://lpsolve.sourceforge.net/5.5/>
2. ILOG CPLEX, ILOG, Inc. (2006), <http://www.ilog.com/products/cplex/>
3. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
4. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., et al.: ProM 4.0: Comprehensive Support for Real Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
5. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering* 47(2), 237–267 (2003)

6. van der Aalst, W.M.P., Rubin, V., van Dongen, B.F., Kindler, E., Günther, C.W.: Process Mining: A Two-Step Approach using Transition Systems and Regions. *Acta Informatica* (submitted, 2007)
7. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. *Knowledge and Data Engineering* 16(9), 1128–1142 (2004)
8. Badouel, E., Bernardinello, L., Darondeau, Ph.: Polynomial Algorithms for the Synthesis of Bounded Nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995)
9. Badouel, E., Darondeau, Ph.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
10. Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? VipTool Tells You! In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 381–390. Springer, Heidelberg (2006)
11. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from Finite Transition Systems. *IEEE Transactions on Computers* 47(8), 859–882 (1998)
13. Darondeau, P.: Deriving Unbounded Petri Nets from Formal Languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998)
14. Desel, J., Esparza, J.: Free Choice Petri Nets. *Cambridge Tracts in Theoretical Computer Science*, vol. 40. Cambridge University Press, Cambridge (1995)
15. van Dongen, B.F.: Process Mining and Verification. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2007)
16. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica* 27(4), 315–368 (1989)
17. Lorenz, R., Juhás, R.: How to Synthesize Nets from Languages - a Survey. In: *Proceedings of the Wintersimulation Conference (WSC) 2007* (2007)
18. Peterson, J.L.: Petri net theory and the modeling of systems. Prentice-Hall, Englewood Cliffs (1981)
19. Reisig, W.: Petri Nets: An Introduction. *Monographs in Theoretical Computer Science: An EATCS Series*, vol. 4. Springer, Berlin (1985)
20. Schrijver, A.: Theory of Linear and Integer programming. Wiley-Interscience, Chichester (1986)
21. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering* 10(2), 151–162 (2003)

Synthesis of Petri Nets from Scenarios with VipTool

Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser*

Department of Applied Computer Science,
Catholic University of Eichstätt-Ingolstadt
`firstname.lastname@ku-eichstaett.de`

Abstract. The aim of this tool paper is twofold: First we show that VipTool [9,2] can now synthesize Petri nets from partially ordered runs. To integrate this extension and further new functionalities, we changed the architecture of VipTool to a flexible plug-in technology. Second we explain how VipTool including the synthesis feature can be used for a stepwise and iterative formalization and validation procedure for business process Petri net models. The synthesis functionalities fill a gap in a previously defined procedure [9,7] where the first definition of an initial process model had to be done "by hand", i.e. without any tool support.

1 Introduction

Automatic generation of a system model from a specification of its behaviour in terms of single scenarios is an important challenge in many application areas. Examples of such specifications occurring in practice are workflow descriptions, software specifications, hardware and controller specifications or event logs recorded by information systems.

In the field of Petri net theory, algorithmic construction of a Petri net model from a behavioural specification is known as synthesis. *Synthesis of Petri nets* has been a successful line of research since the 1990s. There is a rich body of nontrivial theoretical results, and there are important applications in industry, in particular in hardware system design, in control of manufacturing systems and recently also in workflow design.

The classical *synthesis problem* is the problem to decide whether, for a given behavioural specification, there exists an unlabelled Petri net, such that the behaviour of this net coincides with the specified behaviour. In the positive case, a synthesis algorithm constructs a witness net. For practical applications, the main focus is the computation of a system model from a given specification, not the decision of the synthesis problem. Moreover, applications require the computation of a net, whether or not the synthesis problem has a positive answer. To guarantee a reasonable system model also in the negative case, in this paper, the synthesized model is demanded to be the best (a good) upper approximation, i.e. a net including the specified behaviour and having minimal (few) additional behaviour (for details see [12,1]).

Theoretical results on synthesis mainly differ w.r.t. the considered *Petri net class* and w.r.t. the considered model for the *behavioral specification*. Synthesis can be applied to various classes of Petri nets, including elementary nets [10,11], place/transition nets (p/t-nets) [1] and inhibitor nets [13]. The behavioural specification can be given by a

* Supported by the project "SYNOPS" of the German Research Council.

transition system representing the sequential behaviour of a system or by a step transition system additionally considering steps of concurrent events [1]. Synthesis can also be based on a language. Languages can be finite or infinite sets of occurrence sequences, step sequences [1] or partially ordered runs [12]. Despite of this variety, the synthesis methods follow one common theoretical concept, the so called *theory of regions*.

Although there is a wide application field, there is only few tool support for Petri net synthesis so far. Synthesis of labelled elementary nets from transition systems is implemented in the tool Petrify [6]. Petrify is tailored to support hardware system design by synthesizing asynchronous controllers from signal transition graphs (both modelled as Petri nets). The process mining tool ProM [15] uses synthesis methods to automatically generate a process model (given by a Petri net) from so called event logs. Event logs specify a set of occurrence sequences. But ProM does not offer pure Petri net synthesis algorithms. Synet [5] is a software package synthesizing bounded p/t-nets from transition systems. Synet supports methods for the synthesis of so called distributable Petri nets (consisting of distributed components which communicate by asynchronous message passing). It was applied in the area of communication protocol design. However, the authors point out that the tool should be considered as a preliminary prototype rather than a stable and robust tool.

This paper presents a new synthesis tool. The tool supports for the first time synthesis of place/transition-nets from a set of partially ordered runs. The set of partially ordered runs models alternative executions of a place/transition-net. The tool is implemented as an extension of VipTool [9,2], which was originally designed at the University of Karlsruhe within the research project VIP (Verification of Information systems by evaluation of Partially ordered runs) as a tool for modelling, simulation, validation and verification of business processes using (partially ordered runs of) Petri nets. With this new synthesis package all aspects concerned with partially ordered runs of Petri nets – namely synthesis, unfolding (combined with respective validation) and testing of executability (see Figure 1) – are covered by VipTool. So far, the synthesis package is restricted to synthesis from finite sets of partially ordered runs as described in [12]. Typical applications of synthesis only have to deal with finite specifications. Since occurrence sequences and step sequences are special cases of partially ordered runs, the synthesis algorithms are applicable for all kinds of finite languages. Two conceptually different synthesis methods are implemented, covering the major language based synthesis approaches described in the literature [13]. The main approaches to deal with finite representations of infinite languages will be implemented in the near future.

Due to the growth of the number of functionalities of VipTool, we redesigned VipTool in a flexible open plug-in architecture. Algorithms are implemented as plug-ins.

There are many Petri net tools for modelling and analysis, but VipTool occupies a special niche: VipTool is the only tool offering a comprehensive bundle of methods (see Figure 1 for an overview) concerned with causality and concurrency modelled by partially ordered runs of Petri nets (other tools concerned with partial order behaviour of Petri nets focus on model checking techniques employing unfoldings, but not on causality and concurrency). In contrast to occurrence and step sequences, partially ordered runs allow to represent arbitrary concurrency relations between events. Therefore, they are very well suited for the modelling of scenarios of concurrent systems. Advantages

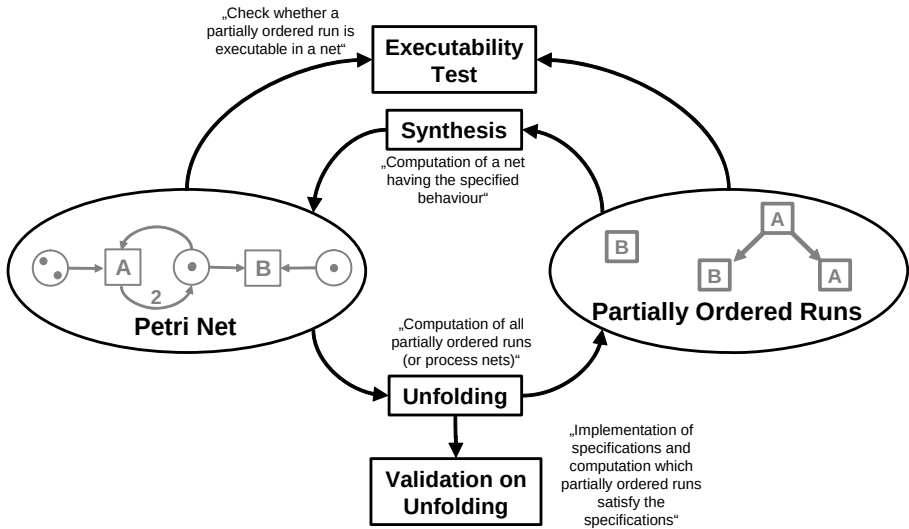


Fig. 1. Sketch of the key functionalities of VipTool

over sequential scenarios are in intuition, efficiency and expressiveness [7,9]. In particular, to specify system behaviour, instead of considering sequential scenarios and trying to detect possible concurrency relations from a set of sequential scenarios, it is much easier and intuitive to work with partially ordered runs.

Starting with a specification of a distributed system in terms of partially ordered runs, the synthesis package of VipTool is used for the generation of prototypes, to uncover system faults or additional requirements using Petri net analysis, to evaluate design alternatives, to visualize a specification or even for the automatic generation of final system models (see [7,8,9,14] for the significance of scenarios in requirements engineering and system design). In the remainder of this section, we explain how to apply the synthesis package in business process design and how the new functionalities work together with the existing validation and verification concepts of VipTool.

One of the main issues of modelling a business process is analysis. Obviously, analysis requires a high degree of validity of the model with respect to the actual business process in the sense that the model faithfully represents the process. For complex business processes, a step-wise design procedure, employing validation of specifications and verification of the model in each step, was suggested in [7,9]. As shown in [2,9], so far VipTool supported most steps of this approach (see also Figure 1). It generates occurrence nets representing partially ordered runs of Petri net models. Specifications can be expressed on the system level by graphical means. Occurrence nets are analyzed w.r.t. these specified properties. Runs that satisfy a behavioural specification are distinguished from runs that do not agree with the specification. The algorithms of VipTool for testing executability of scenarios offer functionalities for detailed validation and verification of the model or a specification w.r.t. single runs. A complicated step that is not supported by previous VipTool versions is the creation of an initial Petri net model for the design procedure. The classical approach to develop a process model is identifying

tasks and resources of the process and then directly designing the control-flow. The validity of the model is afterwards checked by examining its behaviour in comparison to the behaviour of the business process. Using synthesis algorithms (as supported by VipTool) the procedure changes [8]. The desired behaviour of the model constitutes the initial point. First scenarios (in some contexts also called use cases) of the business process are collected. Then the process model is automatically created. In this approach the main task for the user is to design appropriate scenarios of the process (exploiting descriptions of known scenarios that have to be supported by the business process). Usually, it is less complicated to develop and model single scenarios of a process than directly modelling the process as a whole. In particular, in contrast to a complete process model, scenarios need not be designed by some modelling expert, but they may also be designed independently by domain experts.

In Section 2 we survey the new architecture and the features of VipTool. In Section 3 the new synthesis functionalities are illustrated with a small case study.

2 Architecture and Functional Features of VipTool

In this section we provide a detailed description of the functionalities and the architecture of VipTool. VipTool is a platform independent software tool developed in Java. Its previous versions [9,2] were standalone applications providing functionalities for the analysis of partially ordered runs of Petri nets based upon a Java Swing GUI. Due to the growth of functionalities and to increase feature extensibility we redesigned VipTool as an open plug-in architecture. The focus is still on algorithms concerned with partially ordered runs. VipTool uses only standard Java libraries and a GUI developed with the Java swing widget toolkit. It is implemented strictly following advanced object oriented paradigms. In the development of VipTool, we accounted for professional standards such as design patterns, coding conventions, a bug-tracking system and an extensive documentation. The following paragraph gives an overview of the new architecture of VipTool.

The VipTool core platform offers a flexible plug-in technology. The so-called extension manager serves as the foundation for the integration of plug-ins. More precisely, it provides functions to load plug-ins into VipTool. Additionally, the core platform provides definition and implementation of basic GUI elements and related actions, project management functionalities, job scheduling organization as well as system-wide configurations. Development of plug-ins is supported by the VipTool SDK library. VipTool SDK provides appropriate interfaces that can be implemented by plug-ins as well as interfaces arranging the connection to the VipTool core platform. VipTool SDK also includes some key support classes. Embedding of plug-ins into the GUI of the core platform is managed via xml-specifications supporting easy extensibility and scalability. That means, xml-files are used to configure the GUI of VipTool by adequately adding and extending menus, buttons, etc. relating to respective plug-ins. The functional components of VipTool are arranged in an open plug-in architecture connectable to the core platform by the extension manager. Figure 2 depicts this architecture, where the functional component plug-ins are bundled to packages including homogeneous functionalities. Component dependencies are indicated by arrows.

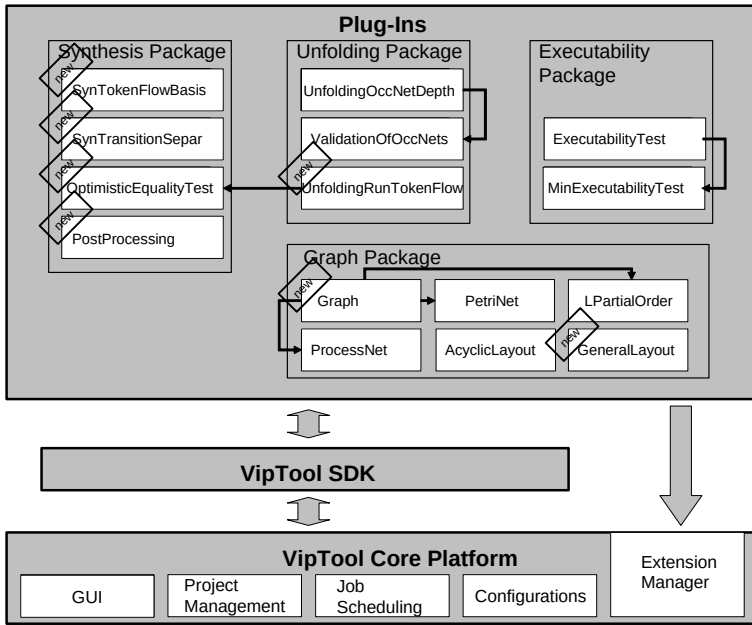


Fig. 2. Architecture of VipTool

The key functionalities of previous VipTool versions have been extracted and reimplemented in independent plug-ins. In Figure 2, new components of VipTool are distinguished from reimplemented functionalities by a "new"-label. Since the number of supported graph and Petri net classes grew, we decided to define graph classes as general as possible in a special plug-in and to implement editing and user interaction functionalities for certain graph classes as additional plug-ins ("Graph Package"). Plug-ins providing algorithmic functionalities ("Synthesis Package", "Unfolding Package", "Executability Package") can be executed in different threads using the job system of the core platform. Plug-ins may communicate with the core platform according to arbitrary communication patterns. The implementation of useful standard communication patterns for algorithmic plug-ins such as status messages, progress bars and logging information is supported by special interfaces of VipTool SDK. Algorithms can manipulate data by using common datastructures defined in Java classes and interfaces of certain plug-ins. To simplify integration of components developed independently or cooperatively by different teams, VipTool also supports easy data exchange between plug-ins and the core platform using xml-files, e.g. pnml-files in the case of Petri nets (pnml is a widely acknowledged standard exchange format for Petri nets) and lpo-files in the case of partially ordered runs (lpo is an xml-file format in the style of pnml). Thus, common datastructures among plug-ins are not required, facilitating extensibility, scalability, composability and reusability of VipTool functionalities. Respective xml-files are arranged by the core platform in workspaces and projects, allowing an arbitrary subfolder structure. The core platform supports a project tree window to offer easy file

management. This is important for flexible and easy invocation of algorithms because the various functionalities have different and partly complex input and output types.

Short descriptions of the VipTool plug-ins shown in Figure 2:

Graph Package

Graph: Provides basic graph classes and interfaces. This plug-in forms the foundation of the "PetriNet", "LPartialOrder" and "ProcessNet" plug-in.

PetriNet: Includes Petri net visualization and editing functionalities as well as simple interactive features such as playing the token game and showing pre- and post-sets.

LPartialOrder: Supports visualization and design of partially ordered runs by labelled partial orders and offers some related functionalities such as computing the transitive closure of a binary relation or the skeleton of a partial order.

ProcessNet: Provides visualization functionalities for occurrence nets.

Acyclic Layout: Offers automatic layout functionalities for directed acyclic graphs such as labelled partial orders and occurrence nets (based on the Sugiyama algorithm).

General Layout: Offers automatic layout functionalities for general graphs such as Petri nets (based on the spring embedder algorithm by Fruchterman and Reingold).

Synthesis Package

SynTokenFlowBasis: Implements the constructive part of the synthesis algorithm for place/transition Petri nets from a finite set of partially ordered runs as described in [4,12]. So called token flow regions and a finite representation by basis regions are applied [13] (employing the algorithm of Chernikova). The result of the algorithm is a net representing a best upper approximation to the specified behaviour.

SynTransitionSepar: Implements the constructive part of a synthesis algorithm for place/transition Petri nets from a finite set of partially ordered runs using a translation to step sequences (based on methods described in [1]). So called transition regions and a finite representation by separating regions are applied [13] (employing the Simplex method). The result of the algorithm is either a negative answer to the synthesis problem combined with a net, illustrating the reason for the negative answer, or a net representing a best upper approximation to the specified behaviour. In the first case the computed net is a good upper approximation but not necessarily a best upper approximation to the specified behaviour (although a best upper approximation exists).

OptimisticEqualityTest: Implements the optimistic equality test described in [4,12] using the newly developed unfolding plug-in "UnfoldingRunTokenFlow" (employs a graph isomorphism test between single partially ordered runs by a branch-and-bound procedure optimized for partially ordered runs [4]). It shows if the behaviour of a net synthesized by the "SynTokenFlowBasis" or the "SynTransitionSepar" plug-in matches the specified behaviour. In the positive case the synthesis problem has a positive answer, otherwise a negative answer.

PostProcessing: Contains a simple and very fast method to delete implicit places from a net. This reduces the size of nets synthesized with the "SynTokenFlowBasis" or the "SynTransitionSepar" plug-in. More advanced post-processing methods are planned.

Unfolding Package

UnfoldingOccNetDepth: Unfolds a Petri net to its occurrence nets (following standard techniques). Occurrence nets are constructed on the fly in a depth first order. Also construction of the branching process including cut-off criteria is supported. See also [9].

ValidationOfOccNets: Allows to specify graphically certain properties of a Petri net, like specific forms of forbidden and desired behaviour. The set of runs, computed by the "UnfoldingOccNetDepth" plug-in, is divided into these runs, which fulfill the specifications, and runs, which do not fulfill at least one of the specifications. See also [9].

UnfoldingRunTokenFlow: Implements the unfolding algorithm to construct the set of all partially ordered runs of a Petri net described in [3]. The algorithm applies an innovative unfolding concept based on token flows, which in the case of general place/transition-nets is considerably superior to standard unfolding algorithms in time and memory consumption. This is in particular important to improve the runtime of the "OptimisticEqualityTest", which depends on an unfolding algorithm. The algorithm does not regard cut-off criteria.

Executability Package

ExecutabilityTest: Supports the polynomial test, whether a given partially ordered run is executable in a given Petri net, explained in [2] (employing methods from the theory of flow networks). The plug-in facilitates failure analysis and constructs an occurrence net corresponding to a checked partially ordered run. See also [2].

MinExecutabilityTest: Offers an algorithm to compute whether a partially ordered run, executable in a Petri net, is minimal executable (based on the plug-in "ExecutabilityTest"). See also [2].

3 Case Study

We briefly illustrate the new synthesis functionalities of VipTool by a simple case study. We consider the workflow caused by a damage report in an insurance company, i.e. how a claim is processed. The workflow is described by possible (alternative) scenarios of the business process represented by partially ordered runs (note that it is enough to consider maximal runs with minimal causality). A Petri net model of the business process is automatically generated either by the "SynTokenFlowBasis" plug-in or the "SynTransitionSepar" plug-in.

Figure 3 shows the partially ordered runs modelled in VipTool and the nets computed with the synthesis plug-ins. There are three possible scenarios: All start with the registration of the loss form submitted by the client (task "Register"), followed by two concurrent tasks "Check Damage" and "Check Insurance". The latter models checking validity of the clients insurance, the former represents checking of the damage itself. Scenario 1 models the situation that both checks are evaluated positively, meaning that the damage is paid (task "Pay Damage") after the two checks. If one evaluation is negative, the company sends a refusal letter. Thus the task "Send Refusal Letter" is either performed after a negative evaluation of the task "Check Damage" (scenario 2) or after a negative evaluation of the task "Check Insurance" (scenario 3).

Combining these three scenarios to a Petri net by synthesis algorithms yields the net `damageReportTF` in the case of the "SynTokenFlowBasis" plug-in (and the "PostPro-

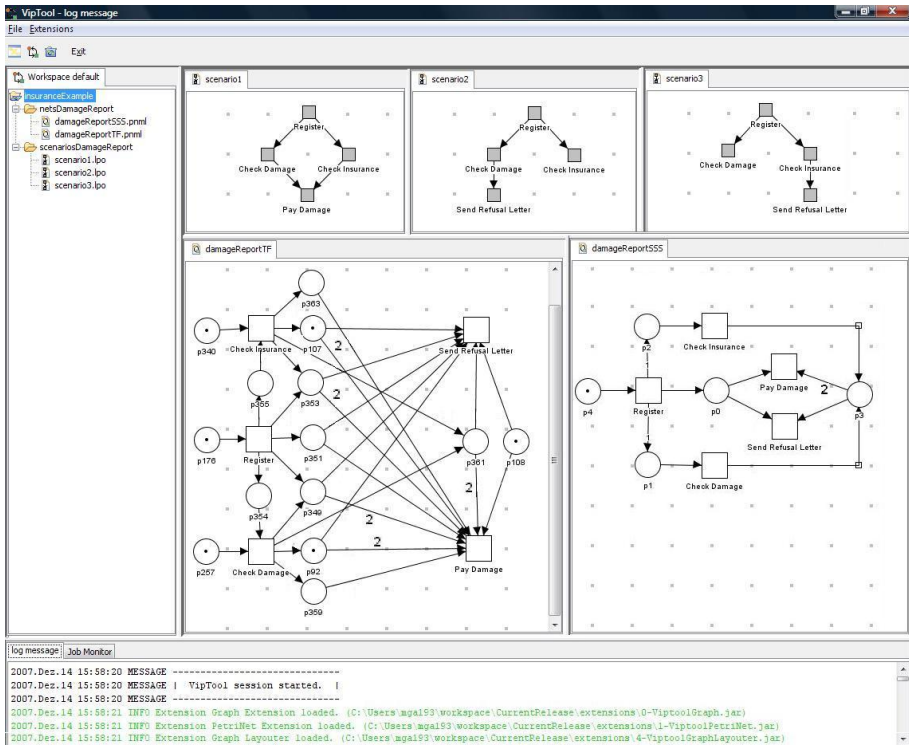


Fig. 3. Screenshot of VipTool showing the standard user interface

cessing” plug-in) and the net damageReportSSS in the case of the ”SynTransitionSepar” plug-in. Both nets faithfully represent the considered business process. The test by the ”OptimisticEqualityTest” plug-in is positive in both cases. While the net damageReportSSS is nearly the simplest Petri net implementation of the considered business process, the net damageReportTF is complicated. This shows that the ”SynTokenFlowBasis” plug-in requires advanced post-processing methods for better readable results.

Figure 4 depicts the scenarios of a more complex variant of the above workflow. The refusal letter can still be sent after the completion of both parallel ”Check” tasks if one is evaluated negatively (scenario 1). But if a negative evaluation of one ”Check” task already causes sending a refusal letter, while the other ”Check” task has not been performed yet, this second ”Check” task has to be disabled (i.e. it does not occur in a respective scenario), since it is no longer necessary (scenario 2 and 3). If both ”Check” tasks are evaluated positively, an acceptance letter is sent (scenario 4-6). Then either the damage is immediately paid (scenario 4) or additional queries to improve estimation of the loss are asked one (scenario 5) or two (scenario 6) times before the damage is paid. Additionally all six scenarios include the task ”Set Aside Reserves”. This task is performed after the ”Register” task in a subprocess concurrent to all other tasks except for ”Pay Damage”. Since the damage is paid from the respective reserves, the reserves

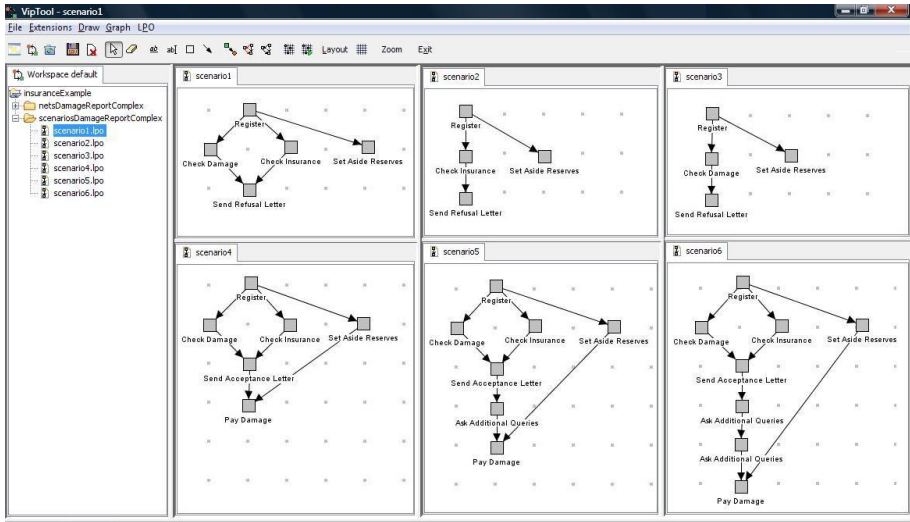


Fig. 4. Screenshot of VipTool showing the user interface of the editor for partially ordered runs

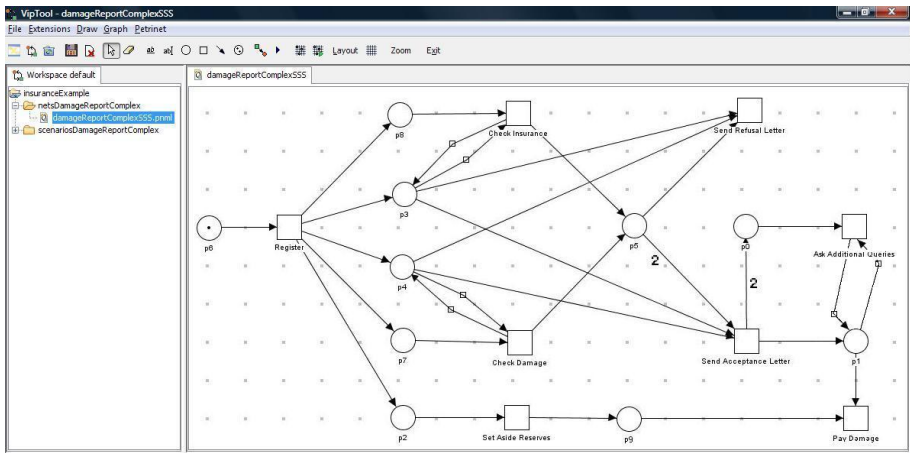


Fig. 5. Screenshot of VipTool showing the user interface of the editor for Petri nets

have to be built up first. Reserves are set aside in any situation, since, in the case the insurance company rejects paying, the reserves have to cover the risk of a lawsuit.

Figure 5 shows the net `damageReportComplexSSS` synthesized with the "SynTransitionSepar" plug-in from the set of partially ordered runs depicted in Figure 4. The net represents a very compact model of the described complex business process. The "OptimisticEqualityTest" yields a positive answer.

The example gives an intuition for our assumption that directly designing a Petri net model of a business process is often challenging, while modelling scenarios and

synthesizing a net is easy. Manually developing a complex Petri net such as the net `damageReportComplexSSS` for the described business process is an error-prone task, but the design of the six scenarios 1-6 is straightforward, yielding automatically a Petri net by synthesis algorithms.

4 Conclusion

In this paper we surveyed the new synthesis package of VipTool and its applicability in business process design. Discussion of the computational complexity of the implemented synthesis algorithms and experimental results (showing the limitations of the algorithms) can be found in [4] and in a journal version of [12] accepted for *Fundamenta Informaticae*. The current version of VipTool can freely be downloaded from "<http://www.ku-eichstaett.de/Fakultaeten/MGF/Informatik/Projekte/Viptool>".

Next steps in the development of VipTool are the implementation of functionalities tuning VipTool to better practical applicability. This includes methods to

- improve the performance of the algorithms of the "Unfolding", "Synthesis" and "Executability" package,
- improve the "PostProcessing" plug-in,
- include further synthesis variants,
- further improve editing functionalities of the "Graph" package,
- deal with high-level nets.

We acknowledge the work of all other members of the VipTool development team: Thomas Irgang, Leopold von Klenze, Andreas Klett, Christian Kölbl, Robin Löscher.

References

1. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
2. Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? Viptool Tells You! In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 381–390. Springer, Heidelberg (2006)
3. Bergenthum, R., Lorenz, R., Mauser, S.: Faster Unfolding of General Petri Nets. In: AWPN 2007, pp. 63–68 (2007)
4. Bergenthum, R., Mauser, S.: Experimental Results on the Synthesis of Petri Nets from Partial Languages. In: Petri Net Newsletter, vol. 73, pp. 3–10 (2007)
5. Caillaud, B.: Synet, <http://www.irisa.fr/s4/tools/synet/>
6. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers. IEE-ICE Trans. of Informations and Systems E80-D(3), 315–325 (1997)
7. Desel, J.: Model Validation - A Theoretical Issue? In: Esparza, J., Lakos, C.A. (eds.) ICATPN 2002. LNCS, vol. 2360, pp. 23–43. Springer, Heidelberg (2002)
8. Desel, J.: From Human Knowledge to Process Models. In: Kaschek, R., et al. (eds.) UNIS-CON 2008. LNBIP, vol. 5, Springer, Heidelberg (2008)
9. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with Viptool. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 380–389. Springer, Heidelberg (2003)

10. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. *Acta Inf.* 33(4), 297–315 (1996)
11. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-Structures: Part I + II. *Acta Inf.* 27(4), 315–368 (1989)
12. Lorenz, R., Bergenthum, R., Desel, J., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. In: *ACSD 2007*, pp. 157–166. IEEE Computer Society, Los Alamitos (2007)
13. Lorenz, R., Juhás, G., Mauser, S.: How to Synthesize Nets from Languages - a Survey. In: *Proceedings of the Wintersimulation Conference (WSC) (2007)*
14. Seybold, C., Meier, S., Glinz, M.: Scenario-Driven Modeling and Validation of Requirements Models. In: *SCESM 2006*, pp. 83–89. ACM, New York (2006)
15. van der Aalst, W.M.P., et al.: ProM 4.0: Comprehensive Support for *real* Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) *ICATPN 2007*. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)

A Monitoring Toolset for Paose

Lawrence Cabac, Till Dörge, and Heiko Rölke

University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, D-22527 Hamburg
<http://www.informatik.uni-hamburg.de/TGI>

Abstract. PAOSE (Petri net-based Agent-Oriented Software Engineering) combines the paradigm of AOSE (Agent-Oriented Software Engineering, see [10]) with the expressive power of Petri nets – reference nets [12] to be more precise. While AOSE is a powerful approach when it comes to designing and developing distributed (agent) applications, it does not address the problems specific to debugging, monitoring, and testing of these applications, i.e. no global state of the system and very dynamic operating conditions. To tackle these problems, two tools have been developed in the context of PAOSE, which are presented in this work.

Firstly, this paper will give a short overview over the interrelated set of tools, which exists already and supports Petri net-based AOSE. The tools are centered around the Petri net-based multi-agent system development and runtime environment RENEW / MULAN / CAPA.

Secondly, MULAN-Viewer and MULAN-Sniffer will be presented in more detail – two tools to address the issues encountered during debugging, monitoring, and testing agent applications. Both tools are first class members of the aforementioned family. The first tool, MULAN-Viewer, deals with the introspection of agents and agent behaviors, while it also offers rudimentary features for controlling the agent-system. The MULAN-Sniffer as the second tool places emphasis on tracing, visualizing, and analyzing communication between all parts of the multi-agent application and offers interfaces for more advanced methods of analysis, such as process mining. Both MULAN-Viewer and MULAN-Sniffer are realized as RENEW plugins that can also be extended by other plugins.

Keywords: reference nets, RENEW, monitoring, testing, debugging, inspection, analysis, multi-agent systems, PAOSE.

1 Introduction

Developers of multi-agent applications – or distributed systems in general – have to cope with many aspects of such systems that increase the difficulty of activities like debugging, monitoring, and testing. Particularly multi-agent applications sport decentralized control, very dynamic operating conditions, and they are inherently complex. Therefore these tasks are hard [17,19].

The main underlying problem is that distributed systems do not allow for a simple definition of their global state, as the state depends for example on the

location of the observing agent. And even if the definition succeeds, gathering the state of a distributed system is far from trivial – leave alone single-stepping through a sequence of states – because it is hard to freeze the system. In addition to this, system access permissions have to be taken into account, e.g. a developer may not have access to all parts of the system even though these very parts trigger problems with the sub-system being developed by him. Concurrency can also lead to errors that only occur in special settings or are not reproducible.

First of all this paper will give a brief overview over the family of interrelated tools that are used for Petri net-based AOSE (for other approaches compare [1] or [16]). Apart from the base set there are tools for designing and creating, as well as tools for debugging, monitoring, and testing¹ multi-agent applications. From the latter set of tools for inspecting the MULAN-Viewer focuses on the inspection of individual agents, their behavior, and the platforms. The MULAN-Sniffer focuses on the communication between agents. Here communication is traced directly at the transport service layer. A third inspection mechanism actually is based on the inspection of Petri net instances at runtime provided by the virtual machine RENEW as explained in Section 2.

MULAN-Viewer and MULAN-Sniffer are implemented as extensible RENEW plugins. Both are able to connect to remote platforms using TCP/IP connections. By this means they support distributed and independent system inspection.

Section 2 will present the interrelated set of tools crucial for our Petri net-based AOSE process. Sections 3 and 4 will introduce the MULAN-Viewer and the MULAN-Sniffer respectively.

2 A Family of Tools to Support Petri Net-Based AOSE

This section will briefly describe the “relatives” of the inspecting tools, which constitute the main focus of this paper. All tools are under constant development and key for the research activities conducted in our group.

2.1 Tool Basis

RENEW [13,14], MULAN [11,18], and CAPA [7] are the *condicio sine qua non* for the entire Petri net-based AOSE process. RENEW can be thought of as both the virtual machine (VM) and the integrated development environment (IDE) for all our Petri net-based development. It features an extensible plugin architecture, which helps in hooking all the members of our tool family together.

Apart from being the VM it also provides an inspection mechanism for Petri net instances at runtime. The simulator allows to view the token game, run in continuous or single-step mode, allows to set locally or globally and statically or dynamically defined breakpoints and has been enhanced to inspect token objects in a UML-like viewer in version 2.1. In the context of testing and debugging of control-flow and data, handling this kind of inspection on a very low level is of great advantage when it comes to agent behavior (protocol nets).

¹ We regard all activities (*debugging, monitoring, testing*) as forms of *inspection*.

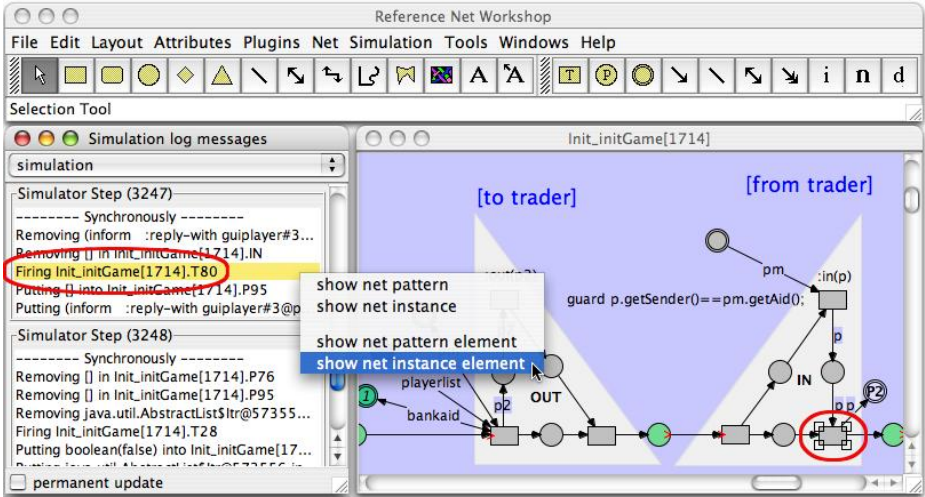


Fig. 1. The IDE part of RENEW showing menu, two palettes (drawing tools and Petri net tools), a part of the simulation log in the log view, and a fragment of a protocol net instance with a highlighted figure (compare with context menu)

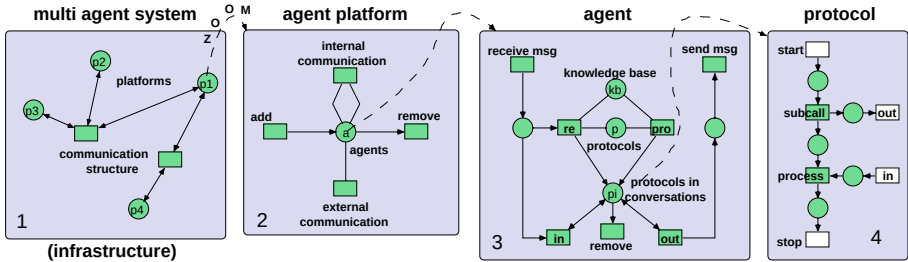


Fig. 2. The MULAN architecture; from infrastructure to agent protocol

Figure 1 shows the graphical interface of RENEW together with a net instance during execution. On the left one can also see the simulation log which permits going through all simulation steps and inspecting the transitions or places.

MULAN describes the reference architecture for multi-agent nets. Strictly speaking an executable conceptual framework. It is heavily based on reference nets and was designed with the corresponding FIPA [8] standards in mind. Figure 2 shows the key architectural approach. CAPA finally is the current reference implementation for MULAN. It is FIPA-compliant and uses Java as well as reference nets (via RENEW) to implement an infrastructure for multi-agent systems.

2.2 Tools for Design and Code Generation

The tools introduced in Section 2.1 have been used numerous times to design large Petri net-based multi-agent applications (MAA). Throughout these

processes several tools were spawned that greatly facilitate the creation of applications, e.g. by automatic code generation (see [6]).

Use Cases. model the relations between actors and their interactions. With the corresponding tool for PAOSE we model agents of the system and their interactions. Thus the diagram provides an overview of the system and we can generate the necessary directory structure for sources of the MAA as well as the file stubs (build files, ontology and knowledge base files). The corresponding palette can be seen in the upper right corner of Figure 3.

Agent Interaction Protocol Diagrams. (AIP) [6] detail the interactions identified during use case (coarse system) design. Tool support consists of rapid modeling support and generation of protocol nets (see menu in Figure 3) to be executed by the agents. The resulting protocol code is made up of directly runnable Petri nets, i.e. reference nets. Figure 3 shows an AIP fragment on the left and a generated agent protocol net (Petri net code) on the right.

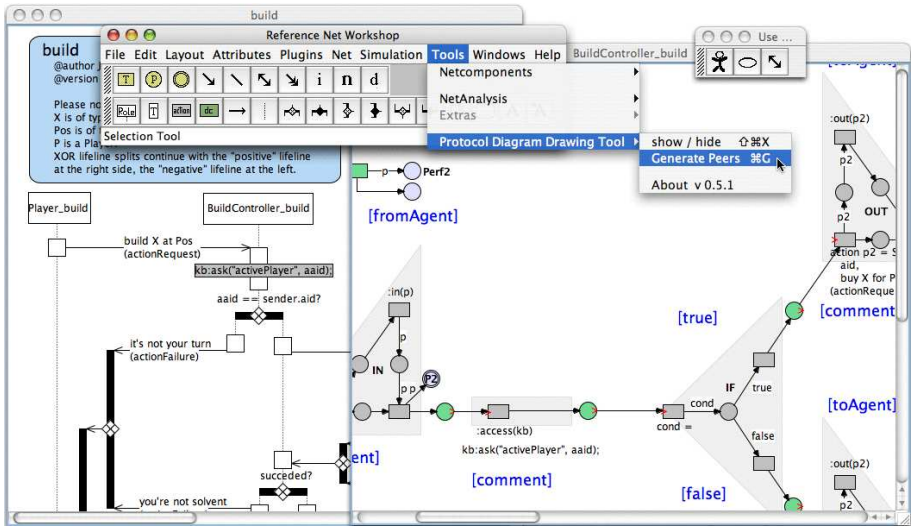


Fig. 3. RENEW menu for generating protocols nets from AIP diagrams

Knowledge Base Editor. [3] is somewhat misleading as a name since it can not only be used to generate agents' knowledge bases. It can also model roles to be assumed by agents, services provided and required by agents as well as the necessary messages. Figure 4 shows the tool with a fragment of the roles and services dependencies modeled in the center. The outline on the right permits selection of specific entries which are displayed on the bottom.

Ontology Generation. [4] provides tool support for creating Java classes that contain the ontology. These classes can then be easily used across the entire multi-agent application.

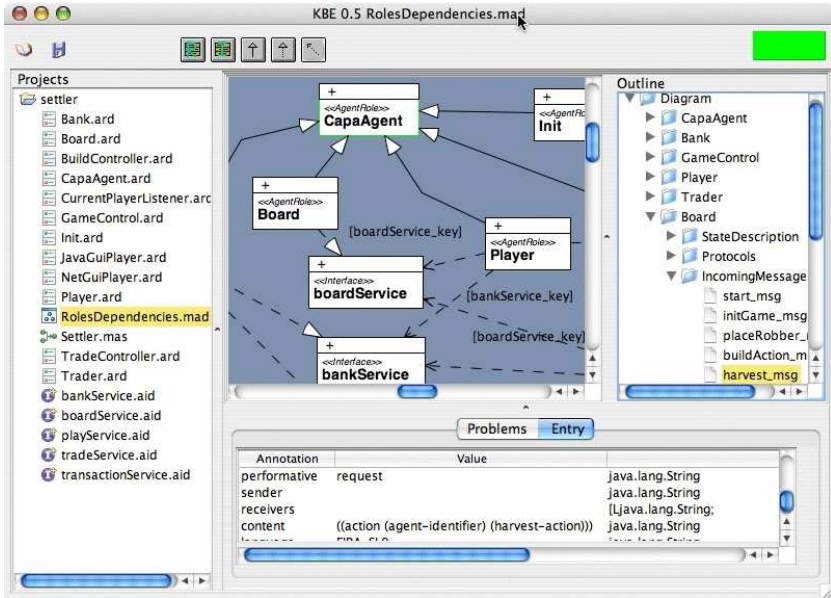


Fig. 4. KBE tool showing modeling of roles and service dependencies

The following sections will present two tools for inspection: MULAN-Viewer and MULAN-Sniffer. It is important to note that both tools have been realized as RENEW plugins and their functionality can be extended by other plugins.

3 MULAN-Viewer

The MULAN-Viewer allows to inspect the state of a multi-agent system (MAS) or several multi-agent systems and its agents, their knowledge bases, and their active protocols and decision components (internal protocols missing external communication features). The main components of the MULAN-Viewer are the platform inspector and the graphical user interface. An arbitrary number of platforms can be inspected both locally and remotely at the same time.

The user interface consists of two views: a MAS overview on the left and the detail view on the right (see Figure 5). The hierarchical structure of the multi-agent system is represented as a tree view. The levels of the tree view correspond directly to three of the four levels known from the MULAN model (see Figure 2): agent platform, agent, and internal agent components (knowledge base, protocols, decision components²). The message transport system agent (MTS) associated with each platform can be seen on the bottom left. If desired, the messages can be listed. All elements can be inspected in the detail view.

² Decision components are omitted in the model, as they are special types of protocols.

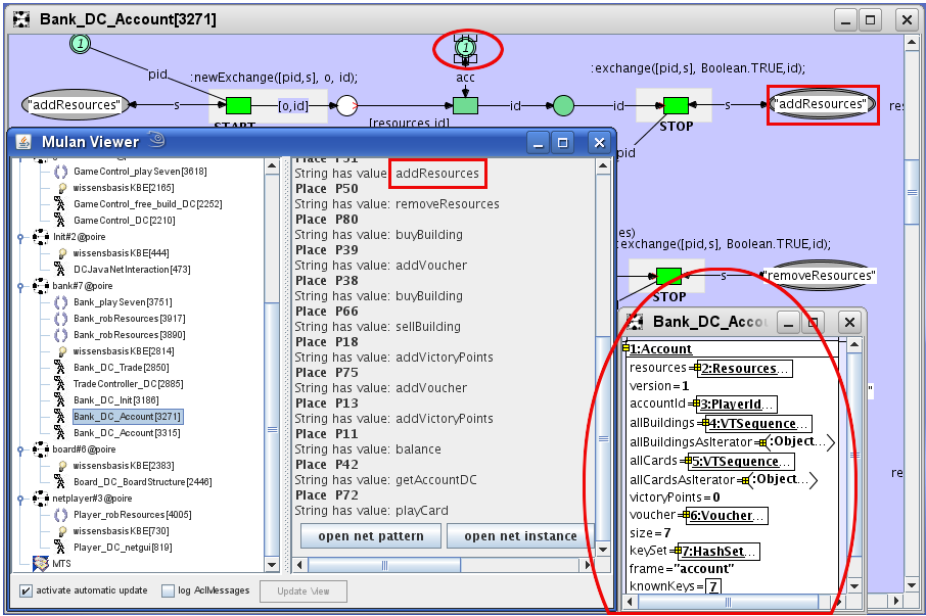


Fig. 5. MULAN-Viewer depicting several agents and their protocols. A net instance (opened from the MULAN-Viewer) can be seen in the background. On the right bottom the content of the place from the top is inspected.

Additionally, underlying Petri nets (as well as Petri net instances) of agents, protocols, decision components and knowledge bases can be directly accessed.

In Figure 5 the running instance of the net *Bank_DC_Account* from agent *bank#7* on platform *poire* has been opened in the detail view (right hand side of the MULAN-Viewer) and can be seen in the background. The superimposed rectangles³ indicate which elements from the detail view correspond to those in the inspected nets. In this case the string *addResources* shown in the detail view of the MULAN-Viewer is contained by the place in question. The parts marked by superimposed ellipses³ show how inspection levels can be nested: First the MULAN-Viewer allows for navigation (left hand side of the MULAN-Viewer) to the desired agent and its protocols. Then the Petri nets can be inspected using RENEW. In the example the place on the top contains one token of type *Account*, which is inspected as UML object hierarchy (*token bag*). The MULAN-Viewer also allows for simple means of control like launching or stopping an agent.

The interaction between the MULAN-Viewer and the agent platform that is being inspected works according to a server/client architecture, with the MULAN-Viewer being the client. The server is realized by an observation module in the platform. More precisely the observation module registers with the underlying simulation engine provided by RENEW. The simulation engine keeps a model of the simulated platform, that can be accessed by modules like the observation

³ Note that the original color of superimposed elements is red.

module. Changes in simulation state are directly updated in the model and propagated through events. The entire communication is based on *Java Remote Method Invocation (RMI)*.

4 MULAN-Sniffer

In every distributed system coordination between the involved sub-systems is important. Coordination usually is accomplished through the exchange of messages. From the developers' points of view it is therefore crucial that they are able to analyze this exchange and inspect the corresponding messages.

For the Petri net-based multi-agent system MULAN / CAPA a suitable tool, the MULAN-Sniffer, has been developed. It was inspired by the JADE sniffer [9]; other related tools and approaches are the ACLAnalyser [2] and the sniffer in MadKit [15]. It uses (agent) interaction protocols (AIP) for visualization and debugging [6,17]. The MULAN-Sniffer focuses on analyzing messages sent by agents in a multi-agent system. The key features are:

portability. The current implementation – realized in *Java* – has been tested with the Petri net-based multi-agent system MULAN / CAPA, but adaption to other FIPA compliant multi-agent systems is easily possible. Theoretically nothing about the target multi-agent system needs to be known, as SL0 content could be directly read from the wire⁴ via libpcap⁵ or comparable means.

modular architecture. Not only the input system is extensible but filtering, analysis and presentation of the messages can be adapted through a powerful plugin system as well. Extensions can even be realized using Petri nets.

distribution. The MULAN-Sniffer is able to gather messages from both local and remote platforms.

filtering. Messages can be selected using stateful or stateless filters. Basic filtering primitives (*from*, *to*, ...) are provided. More sophisticated filters are to be added via the plugin system. Apart from online filtering offline filtering is also possible.

analysis. *Mining-chains* can be used to apply arbitrary analysis algorithms to the messages. Examples are given in [5].

visualization. Apart from showing elementary statistics (total number of messages sent, ...) each message can thoroughly be inspected. Moreover sequence diagrams are auto-generated (in function of the filters applied) on the fly. More complex visualizations can – of course – be realized as plugins. It is interesting to note that the sequence diagrams are actually Agent Interaction Protocol Diagrams. From these AIP Petri net code stubs for agent protocols can be generated again [6]. Thus, agent behavior can be defined by observing a running system turning user interaction (manually) into agent behavior or even allowing the agents to use these observations to adapt their own behaviors.

⁴ Unless cryptography is employed.

⁵ <http://www.tcpdump.org/>, <http://sourceforge.net/projects/libpcap/>

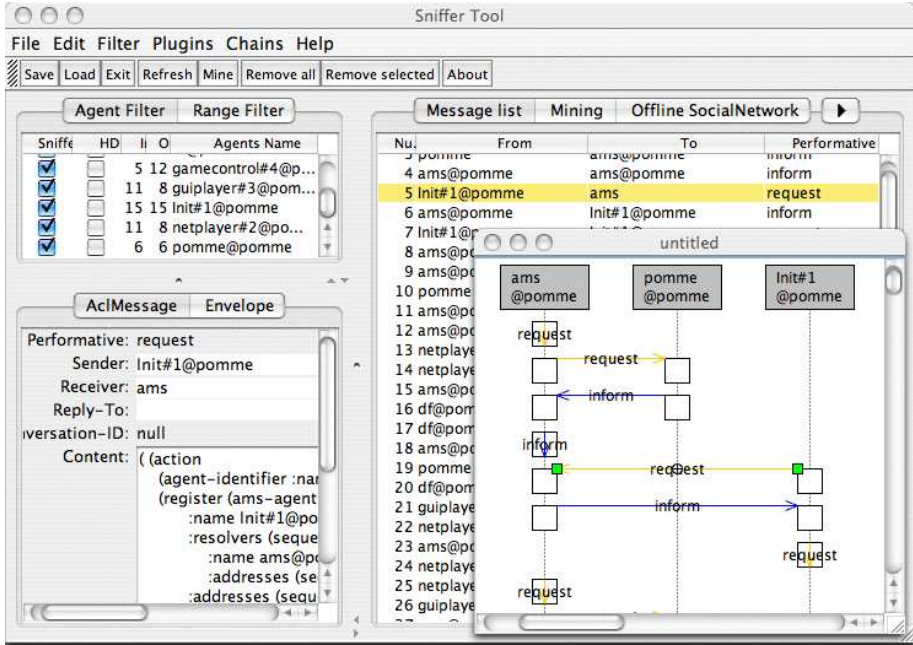


Fig. 6. MULAN-Sniffer main window with generated sequence diagram

Figure 6 shows the MULAN-Sniffer's main window, while sniffing the messages from a teaching project. The main window is divided in three major areas. The top-left one displays the agents known from sniffing the messages. Here, agents can be selected for simple filtering. The right area shows the *Message list*. The currently selected message is displayed in detail in the bottom left area of the main window. Next to the Message list tab one can select from a couple of viewer plugins loaded already. *Online SocialNetwork* (accessible via the arrow next to *Offline SocialNetwork*) for example allows to visualize the frequency of message exchange by pairs of agents. Additionally a part of the on-the-fly auto-generated sequence diagram is shown. Selecting a message arrow in the diagram will highlight the corresponding message in the message list and display the content in the message detail view (tabs: *AclMessage* and *Envelope*) and vice versa.

The MULAN-Sniffer uses the same interface of the platform as the MULAN-Viewer for the collection of messages.

5 Conclusion

In this paper we presented two tools for debugging, monitoring, and testing multi-agent applications. The tools are related with an entire family of tools that form the key elements of our Petri net-based AOSE process. The MULAN-Viewer provides a hierarchical overview of all agents in the system. It also permits

inspecting the state of all elements in the system, whereas the MULAN-Sniffer focuses on the communication of agents and its graphical visualization. Both have been used and enhanced extensively in our teaching projects.

Further possible improvements of the tools are more and better representation of element details in the MULAN-Viewer's interface and increasing the ease of debugging multi-agent applications by adding features for the manipulation of communications (injecting messages), conversations (starting protocols), states (changing knowledge base content), and organizations (migrating agents). To improve the handling of very large scale message logs the integration of an efficient data base is desirable for the MULAN-Sniffer. An interesting topic is the log analysis through advanced techniques such as process mining, for which a prototypical plugin already exists. Further areas of research are ad-hoc systems, support for other platforms, dynamic reconfiguration, and the integration of a security model in the overall architecture as well as the supporting tools.

Acknowledgments

Several authors have contributed to the development of the presented tools. Among these are Timo Carl, Michael Duvigneau, Frank Heitmann, Dr. Daniel Moldt, Florian Plähn, and Jörn Schumacher.

References

1. Al-Shabibi, A., Buchs, D., Buffo, M., Chachkov, S., Chen, A., Hurzeler, D.: Prototyping object oriented specifications. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 473–482. Springer, Heidelberg (2003)
2. Botía, J.A., Hernansaez, J.M., Skarmeta, F.G.: Towards an approach for debugging mas through the analysis of acl messages. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 301–312. Springer, Heidelberg (2004)
3. Cabac, L., Dirkner, R., Rölke, H.: Modelling service dependencies for the analysis and design of multi-agent applications. In: Moldt, D. (ed.) Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA 2006, number FBI-HH-B-272/06 in Reports of the Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany, June 2006, pp. 291–298. University of Hamburg, Department of Informatics (2006)
4. Cabac, L., Dörge, T., Duvigneau, M., Reese, C., Wester-Ebbinghaus, M.: Application development with Mulan. In: Moldt, D., Kordon, F., van Hee, K., Colom, J.-M., Bastide, R. (eds.) Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2007), Siedlce, Poland, June 2007, pp. 145–159. Akademia Podlaska (2007)
5. Cabac, L., Knaak, N., Moldt, D., Rölke, H.: Analysis of multi-agent interactions with process mining techniques. In: Fischer, K., Timm, I.J., André, E., Zhong, N. (eds.) MATES 2006. LNCS (LNAI), vol. 4196, pp. 12–23. Springer, Heidelberg (2006)
6. Cabac, L., Moldt, D.: Formal semantics for AUML agent interaction protocol diagrams. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 47–61. Springer, Heidelberg (2005)

7. Duvigneau, M., Moldt, D., Rölke, H.: Concurrent architecture for a multi-agent platform. In: Giunchiglia, F., Odell, J., Weiß, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 59–72. Springer, Heidelberg (2003)
8. Foundation for Intelligent Physical Agents (FIPA) – homepage. Foundation for Intelligent Physical Agents, <http://www.fipa.org/>
9. The Sniffer for JADE. Online documentation (January 2008), <http://jade.cselt.it/doc/tools/sniffer/index.html>
10. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence 117(2), 277–296 (2000)
11. Köhler, M., Moldt, D., Rölke, H.: Modelling the structure and behaviour of Petri net agents. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 224–241. Springer, Heidelberg (2001)
12. Kummer, O.: Introduction to Petri nets and reference nets. Sozionik Aktuell 1, 1–9 (2001)
13. Kummer, O., Wienberg, F., Duvigneau, M.: Renew – the Reference Net Workshop. Release 2.1 (May 2006), <http://www.renew.de/>
14. Kummer, O., Wienberg, F., Duvigneau, M., Schumacher, J., Köhler, M., Moldt, D., Rölke, H., Valk, R.: An extensible editor and simulation engine for Petri nets: Renew. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 484–493. Springer, Heidelberg (2004)
15. MadKit (January 2008), <http://www.madkit.org>
16. Mans, R.S., van der Aalst, W.M.P., Bakker, P.J.M., Moleman, A.J., Lassen, K.B., Jørgensen, J.B.: From requirements via colored workflow nets to an implementation in several workflow systems. In: Proceedings of the International Workshop on Coloured Petri Nets (CPN 2007), October 2007, Computer Science Department, Aarhus University (2007)
17. Poutakidis, D., Padgham, L., Winikoff, M.: Debugging multi-agent systems using design artifacts: the case of interaction protocols. In: AAMAS, pp. 960–967. ACM, New York (2002)
18. Rölke, H.: Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen. Agent Technology – Theory and Applications, vol. 2. Logos Verlag, Berlin (2004) (German)
19. Van Liedekerke, M.H., Avouris, N.M.: Debugging multi-agent systems. Information and Software Technology 37, 103–112 (1995)

Animated Graphical User Interface Generator Framework for Input-Output Place-Transition Petri Net Models

João Lourenco and Luís Gomes

Universidade Nova de Lisboa / UNINOVA, Portugal
jpl114084@fct.unl.pt, lugo@uninova.pt

Abstract. The paper presents a development framework allowing the automatic generation of animated graphical user interface associated with specific embedded system controllers, and allowing the association of the characteristics of its behavioral model with specific characteristics of the graphical user interface through a set of dedicated rules. The behavioral model of the system is specified by means of an IOPT (Input-Output Place-Transition) Petri net model, which is represented using a PNML (Petri Net Markup Language) notation. Two main tools are described: the "Animator", and the "Synoptic". The first one is the development environment and allows the definition of the graphical characteristics of the synoptic, and their association with dynamic and static characteristics of the Petri net model (the "Animator"); this tool also supports hierarchical structuring of the synoptic and definition of platform dependent code to link the physical world to the Petri net model signals and events. The second one is responsible for the execution part including embedded control and on-line animation of the graphical user interface (the "Synoptic"). Current main usage of the tool is to support teaching of Petri nets and their application to automation and embedded systems design; yet, application as a SCADA (Supervisory, Control, and Data Acquisition) system is envisaged. The application of the tools to a Petri net model of a parking lot controller is briefly presented.

1 Introduction

Several models of computation have been widely accepted for embedded systems behavioral modeling. When facing the implementation of that kind of systems, very often one needs to produce a graphical user interface that could present in a comprehensive way the status of the process under control. For that end, we often need a graphical user interface to present the synoptic of the process under control (for synoptic we mean the graphical representation of the status of a system). Several frameworks have been widely accepted for that purpose based on a dataflow modeling and a visual programming interface; this is the case for LabView and MatLab/Simulink frameworks. Also, a plenty of SCADA (Supervisory, Control, and Data Acquisition) systems are available to manage

the monitoring of distributed processes. When coming to discrete-event modeling, and in particular to Petri nets modeling, it is common to find a lack of tools to fully support all phases of the engineering development process, namely code generation and integration with animated and/or interactive graphical user interface and SCADA systems.

The goal of this paper is to present a design automation tool amenable to generate a graphical user interface application associated with the monitoring of a specific process. The behavioral model of the process controller is expressed through a Petri net model and is also executed by the graphical user interface application. In other words, the design automation tool described in this paper automatically generates a (centralized) SCADA system that implements the supervision and control of a local graphical user interface based on the execution of a Petri net model.

The paper is structured as follows. The next section presents the proposed architecture and main characteristics for the developed design automation tool. In the following sections a brief description on tool usage and application to a specific example are presented. Before concluding, a brief comparison with related works is produced and the main challenges for future extensions are presented.

2 Proposed Architecture

The goal of the design automation tool to be described is to generate an animated and interactive graphical user interface associated with a specific process, to which one already has available the behavioral model expressed through a Petri net model.

In general terms, the graphical user interface should support the creation of one or more layouts, hierarchically organized if adequate, adding background images and animated images into the respective child windows. The background image reflects whole or partial view of the simulated environment (for instance, the whole perspective of a parking lot, or of a building plant). This functionality allows monitoring different parts of the process and associated Petri net model dependencies in different windows (or monitors). Other key functionality allows the addition to the layouts of images and animated images as well (superimposed to the background). These images and animated images can be resized and moved into different locations, and can also be chosen if the designer of the graphical interface wants transparency around the image or not, so the pictures can have arbitrary shapes (not only squares and rectangles). Finally, the interaction with the world, at current stage of developments, is accomplished through common in- and output devices, as mouse, keyboard and monitor.

Fig. 1 presents the proposed architecture for the development framework, showing the interaction of the tool, named "Animator", with other key applications, namely a Petri net graphical editor, an automatic code generator from PNML to C, and the generated graphical user interface.

The flow starts at the bottom left of the figure, when a graphical Petri net editor is used to produce the behavioral model of the system [1], named as

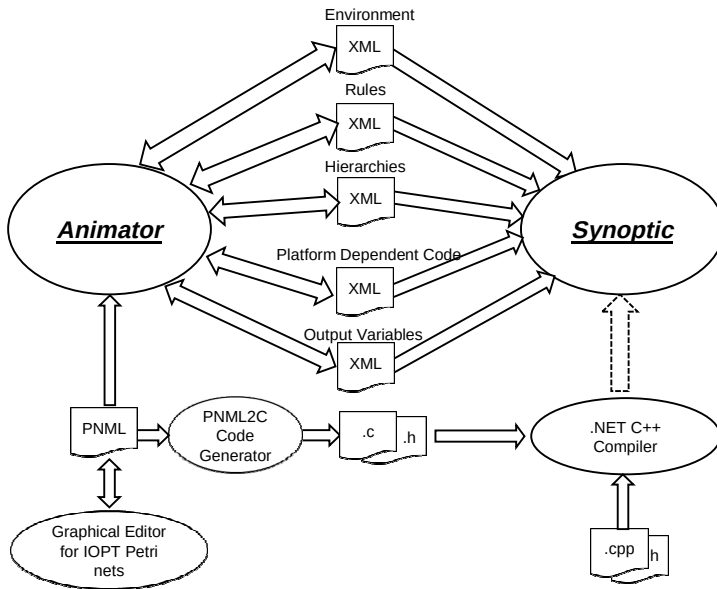


Fig. 1. System architecture

Snoopy-IOPT. This graphical editor is built upon the core code of Snoopy [2], which was kindly provided by the Group of Data Structures and Software Dependability at Brandenburg University of Technology Cottbus, Germany. The Snoopy-IOPT graphical editor is developed supporting a specific Petri net extension, named as IOPT nets (Input-Output Place-Transition Petri nets models), and is able to produce a PNML representation of the model (PNML is an XML-based interchange format for all Petri net classes [3,4]). The IOPT nets were proposed elsewhere [5], and are based on Place-Transition nets, augmented with signals and events modeling. The events and signals allow the specification of interactions between the controller model (the net) and the environment, as in interpreted and synchronized nets. Additionally, IOPT nets also allow the specification of priorities in transitions, and the use of test arcs. Both contribute to integrate conflict resolution arbiters into the models, allowing deterministic execution of the model, which is a key feature whenever controller modeling is concerned.

A PNML to C translator tool is used to produce implementation code in ANSI C language [6] associated with the specific model at hand. Afterwards this code is linked with a set of classes coded in C++ (prepared to implement the desired presentation and interaction with the graphical user interface application), using the .NET C++ compiler framework, and producing as final result the "Synoptic" application. The .NET C++ framework proved to be adequate for the construction of this "proof-of-concept" application, as far as it provides basic infrastructure for development of the type of envisaged graphical interfaces.

The role of the "Animator" tool is to prepare the configuration files to be read at run-time by the "Synoptic" application. These files contain information on all relevant aspects of the interface, including the graphical characteristics and interactivity support, taking into account the information about the IOPT model stored in the PNML file. The "Animator" application has five main outcomes, which will be briefly presented in the following sub-sections and are associated with the five key characteristics of the "Synoptic" application: the set of rules associating Petri net model characteristics and graphical features ("Rules"), the definition of the overall graphical layout of the graphical user interface ("Environment"), the dependency between different graphical layers ("Hierarchies"), the way how to connect input signals and events from the physical world ("Platform Dependent Code"), and finally the way how to present output variables (signals, events, transition firing or place marking) ("Output Variables"). These configuration files are stored using XML format (in order to allow easy interoperability with other tools). In the following subsections, associated main characteristics of the five referred aspects are briefly described.

Animation rules characterization. One central conceptual aspect of the tool relies on the association of characteristics of the Petri net model with specific graphical attributes of the graphical user interface. This is accomplished using a set of rules, as presented in Fig. 2, where the antecedents of the rules reflects dependencies on static characteristics (marking and signals) and dynamic characteristics (transitions and events) of the Petri net model.

In this sense, the graphical user interface can be automatically updated on the occurrence of a change on the model state (place marking, transition firing, signals and events). Consequents of the rules can accommodate several actions, namely showing/hiding of an image, showing/hiding of an animated image, apply a zooming to an image/animated image, define a path where an image should move along (with or without a timer), and present an output variable.

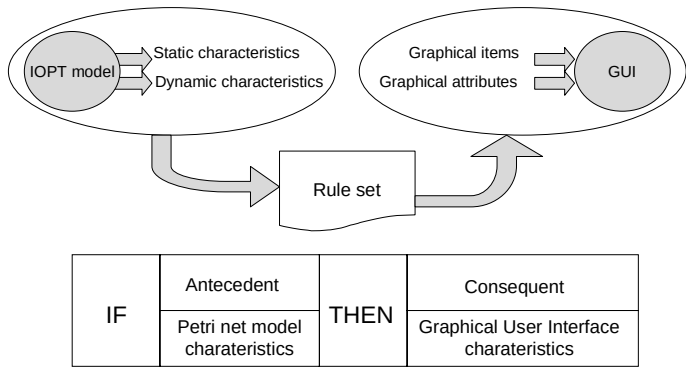


Fig. 2. Associating Petri net model characteristics with specific graphical user interface characteristics

Rule ID	IF	==	THEN	Location	Zoom(%)	Visible	Time(s)
0	Transitions:T1:2293	1					
0	Places:P3:2415	>=3	object	{x=10;10}		True	5
0			object	{x=50;50}			10

Fig. 3. Defining a rule

Fig. 3 presents a snapshot of the rule editor where the antecedents are "transition T1 fires and place P3 holding three or more tokens" and the consequents specify moving of the graphical object "object" from current position to position {10,10} in 5 seconds, followed by a second movement to position {50,50} in 10 seconds.

Environment characterization. The environment characterization accommodates representation of general characteristics of the graphical user interface, namely number of type of child windows, background images, associated images and animated images.

GUI hierarchical structuring. In order to manage graphical user interface complexity, namely different windows to show or to hide at a specific point in time, some basic navigation capabilities are available. This includes three types of navigation: Hierarchical relation (allowing navigation from the parent window to any of the child windows); Unidirectional functional relation (allowing navigation from one window to another one); and Bidirectional functional relation (allowing navigation between both windows).

Platform dependent characteristics. The platform dependent characteristics specifies the way an external signal or event is represented inside the graphical user interface. As the Windows OS based PC implementation platform was selected, these characteristics include association of external signals and events to keyboard strokes and interface button activated through mouse clicks. The activation/deactivation of a signal can be made impulsive (only on change), or stable (like a switch).

Output variables definition. Finally, the characterization of the output variables allows the creation of variables that are linked to an output of the Petri net. Those output variables can be shown in a sidebar or inside a child window.

3 Motivating Example and Development Flow

This section briefly presents application to a simple example where one wants to produce a graphical user interface for a controller of a parking lot composed by three floors, two entrances, two exits and four ramps, as shown in Fig. 4.

At each entry zone exists a car entry detector that generates an arrival signal. This signal can generate two input events, one associated with arrival signal's

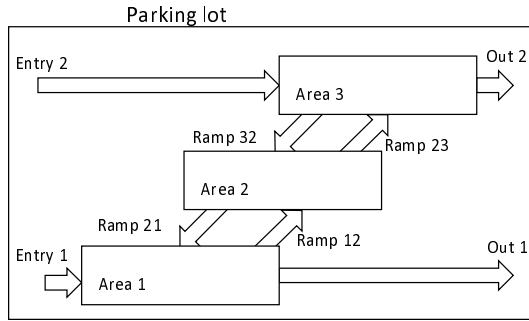


Fig. 4. Layout for a parking lot with three floors, two entrances, two exits, and four ramps between floors

rising edge, and the other one associated with its falling edge. After a car stops at the entry zone and the driver picks up the ticket, the entry gate will open (if some parking place is free) till the car is present at the entrance; afterwards, the gate will close. The same behavior applies to the parking lot exit (with a different set of signals).

This motivating example will also be used to introduce main steps of the development flow allowing a user to start from his/her IOPT Petri net model and end up with the graphical user interface animated by the on-line execution of that IOPT model.

The first step of the development flow is to edit the IOPT model of the system using the Snoopy-IOPT graphical editor. For the motivating example, due to system complexity, an hierarchical representation of the IOPT model was produced, as presented in Fig. 5. Yet, in order to generate the associated C code for execution, the flat PNML file was produced by the Snoopy-IOPT graphical editor. Afterwards, a tool for automatic C code generation from PNML representation will be used.

In this sense, the first group of steps will assure that one will have a PNML representation of the IOPT model of the controller, and the associated C code for execution.

At this point in time, the "Animator" tool will be used to produce configuration files to be ready to be used by the "Synoptic" application. Roughly, the different steps that the designer has to follow will assure the definition of different graphical user interface characteristics, as presented in previous section, and including files for defining "Rules", "Environment", "Hierarchies", "Platform Dependent Code", and "Output Variables".

After this second group of steps, one will be in position to compile the "Synoptic" application and start the simulation of the system through the animated graphical user interface. For our parking lot application, the "Synoptic" looks like in Fig. 6, where several areas can be identified:

- Top-level window presenting an overall graphical user interface of the parking lot, showing the number of cars presented in each floor, as well as the cars

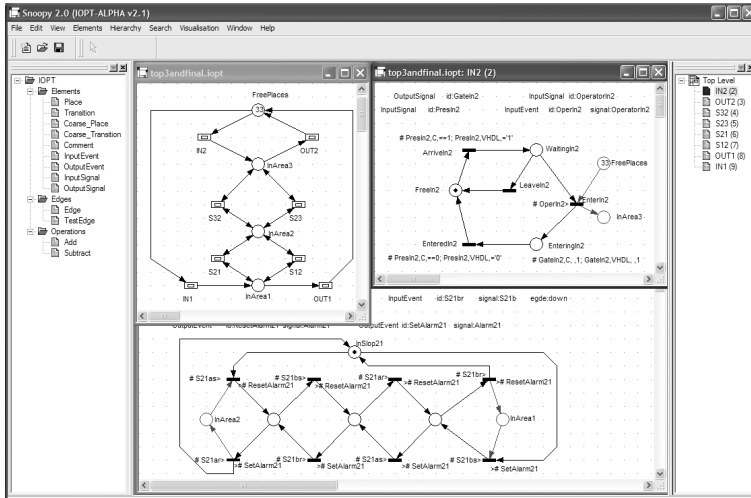


Fig. 5. Hierarchical (partial) representation of the model of the parking lot controller: top-level model (top left), one entrance model (top right), and one ramp model (bottom)

moving in and out, and between floors as well (the definitions of the graphical characteristics of the window are stored in the "Environment" file, while the rules governing the animation of the model are stored in the "Rules" file);

- Bottom window presenting a specific synoptic for area number 3;
- Sidebar (at right hand side) composed by three groups:

Top group – identifying the windows where the user can navigate from the current situation (stored in the "Hierarchies" file); this will allow the user to change the views of the system under control, enabling to change from seeing area 3 to area 2 or area 1, for instance;

Middle group – allowing activation of input signals and events through mouse clicking (stored in the "Platform Dependent Code" file); this area, complemented by keyboard, will allow user to interact with the application;

Bottom group – presenting several output variables actual values (stored in the "Output Variables" file); this area will give visibility to some characteristics of the Petri net model that are not directly visualized in other parts of the synoptic.

The "Synoptic" architecture relies in three main timers. The first one addresses rule execution (rules for animated images and output variables). This timer is also responsible to call functions to manage the queues for animated images that contain multiple paths to go. The second timer is related with what was concluded on the previous timer: executes the rules that have images associated with journeys, producing movements and changes in the images that were defined in the functions related to the previous timer. Finally, the third timer is

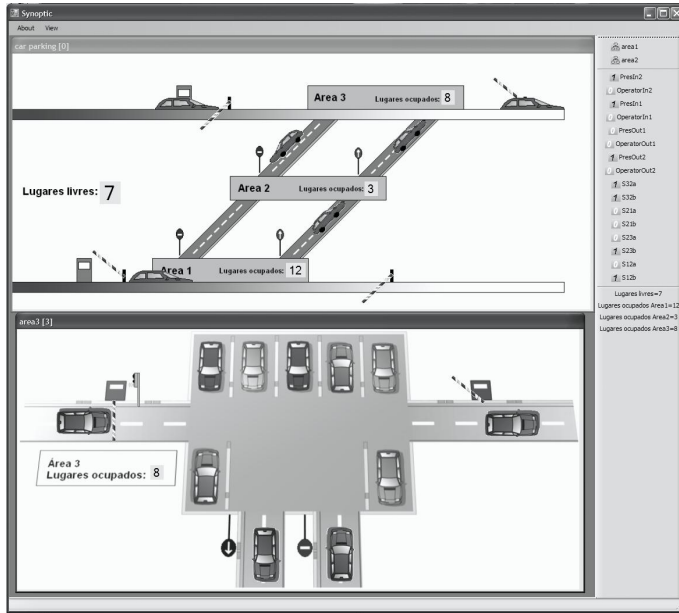


Fig. 6. Generated Graphical user interface for a three floors, 2-entrances, 2-exits parking lot

responsible for updating all images present in all windows, so the user can see the changes made in the procedures called by the second timer.

Files and executable associated with the presented example (as well other examples) are available from the tool webpages at FORDESIGN project website [7].

4 Related Work

Interactive animations based on Petri net models are the goal of several tools already available from the community. Among them, the BRITNeY suite [8,9] is probably the most widely known. It is strongly linked with CPN Tools environment [10], and is based on the addition of code segments to the CPN model transitions, which when fired trigger some actions on the animated application. It has been used worldwide; of special interest, one can refer to one specific application to workflow requirements validation [11].

Several other tools could be considered whenever one needs to face model animation, namely ExSpect [12] (also based on CP-nets), LTSA [13], and PNVis [14] (linked with Petri Net Kernel [15]).

However, one key difference between those solutions (except PNVis) and the "Animator" tool needs to be stressed: the "Animator" tool is able to produce an animated graphical user interface associated with a Petri net model without writing actual code, as the user is guided through a set of interactive dialogues

in order to produce configuration files (design automation tool). In this sense, the "Animator" tool was thought as a "closed" tool (although, it is also possible to include additional code to the animated graphical user interface, of course), supporting an easy and quick solution to set up an animated graphical user interface. So, it is not necessary to have specific programming skills in order to get the animated graphical user interface ready.

5 Conclusion

The "Animator" tool, as well as a set of examples and related documentation are freely available through the FORDESIGN project website [7]. Also the other tools used in cooperation with the "Animator" tool (the Petri net graphical editor and the translator to C) are freely available through the same website.

The potential innovative contributions of the tool described in this paper at current stage of development (as available at the FORDESIGN website [7]) can be summarized as follows: Supporting the validation of the correctness of a model through an interactive simulation based on a synoptic; Being used to support Petri nets teaching, either inside lab classes, or when students are learning on their owns.

Having in mind the usage of the tool by students, it is important to emphasize that the tool can adequately complement other types of simulation (namely the token-player simulation, and simulations of model evolution over time), giving to the student an integrated view on the net model status and its relationship to the process under control. Having in mind the usage of the tool by engineers in order to automatically generate SCADA graphical user interface, it is important to consider new developments on the referred extensions on in- and output connectivity, in order to allow full exploitation of the tool.

Acknowledgment

The authors acknowledge the comments received from anonymous reviewers of the initial version of the paper, and also the comments received from Ekkart Kindler, from Technical University of Denmark. The authors also acknowledge the collaboration received from other members of the group, also tool developers, namely Ricardo Nunes (Snoopy-IOPT Petri net editor) and Tiago Rodrigues (PNML2C tool). This work is supported by the FORDESIGN project (<http://www.uninova.pt/fordesign>), sponsored by Portuguese FCT and POS Conhecimento (FEDER), ref. POSC/EIA/61364/2004.

References

1. Nunes, R., Gomes, L., Barros, J.: A Graphical Editor for the Input-Output Place-Transition Petri Net Class. In: Proceedings of the 2007 IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2007). IEEE, Los Alamitos (2007)

2. S N O O P Y 's home page: Data Structures and Software Dependability – Brandenburg University of Technology Cottbus (2007),
<http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html>
3. Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)
4. PNML: Petri Net Markup Language (PNML) (2004),
<http://www.informatik.hu-berlin.de/top/pnml/about.html>
5. Gomes, L., Barros, J., Costa, A., Nunes, R.: The Input-Output Place-Transition Petri Net Class and Associated Tools. In: 5th IEEE International Conference on Industrial Informatics (INDIN 2007) (2007)
6. Gomes, L., Barros, J.P., Costa, A.: Petri Nets Tools and Embedded Systems Design. In: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2007) (2007)
7. FORDESIGN project: FORDESIGN project home page (2007),
<http://www.uninova.pt/fordesign>
8. Westergaard, M., Lassen, K.B.: The BRITNeY Suite Animation Tool. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 431–440. Springer, Heidelberg (2006)
9. Westergaard, M.: BRITNeY suite website, <http://wiki.daimi.au.dk/tincpn/>
10. CPN-Tools: CPN Tools homepage, <http://wiki.daimi.au.dk/cpntools/>
11. Machado, R.J., Lassen, K.B., Oliveira, S., Couto, M., Pinto, P.: Execution of UML models with CPN Tools for workflow requirements validation. In: Proceedings of Sixth CPN Workshop, DAIMI, vol. PB-576, pp. 231–250 (2005)
12. ExSpect tool: The ExSpect tool website, <http://www.exspect.com/>
13. Magee, J., Kramer, J.: Concurrency - State Models and Java Programs. John Wiley & Sons, Chichester (1999)
14. Kindler, E., Pales, C.: 3d-visualization of Petri net models: Concepts and visualization. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 464–473. Springer, Heidelberg (2004)
15. Weber, M., Kindler, E.: The Petri net kernel. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 109–123. Springer, Heidelberg (2003)

HYPENS: A Matlab Tool for Timed Discrete, Continuous and Hybrid Petri Nets

Fausto Sessego, Alessandro Giua, and Carla Seatzu

Dip. Ingegneria Elettrica ed Elettronica, Università di Cagliari, Italy
{fausto.sessego,giua,seatzu}@diee.unica.it

Abstract. HYPENS is an open source tool to simulate timed discrete, continuous and hybrid Petri nets. It has been developed in Matlab to allow designer and user to take advantage of several functions and structures already defined in Matlab, such as optimization routines, stochastic functions, matrices and arrays, etc. The tool can also be easily interfaced with other Matlab programs and be used for analysis and optimization via simulation. The large set of plot functions available in Matlab allow one to represent the results of the simulation in a clear and intuitive way.

Keywords: Computer tools for nets, Timed and stochastic nets, Hybrid nets.

1 Introduction

In this paper we present a Matlab tool for the simulation of timed Petri nets (PN), called *HYPENS* (HYbrid PEtri Nets Simulator) to emphasize that it deals not only with *discrete* nets, but with *continuous* and *hybrid* nets as well.

In many applications dealing with complex systems, a plant has a discrete event dynamics whose number of reachable states is typically very large, and problems of realistic scale quickly become analytically and computationally untractable. To cope with this problem it is possible to give a continuous approximation of the "fast" discrete event dynamics by means of *continuous Petri nets*, i.e., nets obtained from discrete nets by "fluidification" [1,2].

In general, different fluid approximations are necessary to describe the same system, depending on its discrete state. Thus, the resulting models can be better described as *hybrid Petri nets* (HPN) that combine discrete and continuous dynamics [1,3]. Several HPN models have been defined (see [3]). The model considered in HYPENS is called *First-Order Hybrid Petri nets* (FOHPN) because its continuous dynamics are piece-wise constant. FOHPN were originally presented in [4] and have been successfully used to model and analyze manufacturing systems [5].

In the last years several tools for the simulation of timed discrete PN have been proposed. Very few tools on the contrary, also deal with hybrid PN: we are aware of *HISim* [6] and *SIRPHYCO* [7].

SIRPHYCO is the most complete in terms of modeling power and performance analysis, but still it presents the following limitations. (a) It has a limited number

of modeling primitive. As an example: only stochastic discrete transitions with exponential firing delays are considered; it only assumes infinite-server semantics, thus if we want to consider a finite server semantics we have to add dummy places that complicate the model. (b) It uses a "reserved marking policy" ([1]), i.e., as soon as a discrete transition is enabled, the tokens necessary for its firing are reserved and cannot be used to enable other transitions. In other words, conflicts are solved at the enabling time not at the firing time. (c) Conflict resolution, both between discrete and continuous transitions, are only solved by priorities. (d) The input graphical interface is not practical for large nets, and only a few statistics can be computed from the simulation run. (e) It is not an open source software and cannot be easily interfaced with other programs.

HYPENS overcomes the above limitations, and presents several other advantages. (a) It has many useful modeling primitives such as: finite/infinite server semantics, general stochastic firing delays for discrete transitions. (b) It does not use reserved marking but the tokens in a place can concurrently enable several transitions. In other words, conflicts are solved at the firing time: this policy can be shown to be more general than based on reserved marking. (c) It offers several conflict resolution policies as we will discuss later. (d) It uses a textual input interface but provides several statistical data both numerically and graphically. (e) It is an open source software written in Matlab, to allow designer and user to take advantage of several functions and structures already defined in Matlab and to easily interface it with other programs.

A final remark concerns the issue of *conflict resolution* that is fundamental in any PN simulator [8]. For discrete transitions we use a general approach that combines both priorities and random weighted choices as in [8]: this is coded in the structure of the net. In the case of continuous transitions, on the contrary, we assume that the choice of the firing speed vector $\mathbf{v}$ is made at run-time and represents a control input. Thus, for solving conflicts among continuous transitions we use a more general optimization rule: at each step the net evolves so as to (myopically) optimize a linear performance index $J(\mathbf{v})$.

The software, manual and demos of *HYPENS* can be downloaded from [9].

2 Hybrid Petri Nets

We recall the FOHPN formalism used by *HYPENS*, following [4].

Net structure: A FOHPN is a structure $N = (P, T, Pre, Post, \mathcal{D}, \mathcal{C})$.

The set of *places* $P = P_d \cup P_c$ is partitioned into a set of *discrete* places P_d (represented as circles) and a set of *continuous* places P_c (represented as double circles). The cardinality of P , P_d and P_c is denoted n , n_d and n_c .

The set of *transitions* $T = T_d \cup T_c$ is partitioned into a set of discrete transitions T_d and a set of continuous transitions T_c (represented as double boxes). The cardinality of T , T_d and T_c is denoted q , q_d and q_c .

The *pre-* and *post-incidence functions* that specify the arcs are (here $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$): $Pre, Post : P_c \times T \rightarrow \mathbb{R}_0^+$, $P_d \times T \rightarrow \mathbb{N}$. We require that $\forall t \in T_c$ and

$\forall p \in P_d$, $Pre(p, t) = Post(p, t)$, so that the firing of continuous transitions does not change the marking of discrete places.

Transitions in T_d may either be deterministic or stochastic. In the case of deterministic transitions the function $\mathcal{D} : T_d \rightarrow \mathbb{R}_0^+$ specifies the timing associated to timed discrete transitions. In the case of stochastic transitions $\mathcal{D}$ defines the parameter(s) of the distribution function corresponding to the timing delay. The function $\mathcal{C} : T_c \rightarrow \mathbb{R}_0^+ \times \mathbb{R}_\infty^+$ specifies the firing speeds associated to continuous transitions (here $\mathbb{R}_\infty^+ = \mathbb{R}^+ \cup \{\infty\}$). For any continuous transition $t_j \in T_c$ we let $\mathcal{C}(t_j) = [V_j', V_j]$, with $V_j' \leq V_j$: V_j' represents the *minimum firing speed* (mfs), V_j represents the *maximum firing speed* (MFS).

The *incidence matrix* of the net is defined as $C(p, t) = Post(p, t) - Pre(p, t)$. The restriction of $\mathbf{C}$ (Pre , $Post$, resp.) to P_x and T_y , with $x, y \in \{c, d\}$, is denoted $\mathbf{C}_{xy}$ (Pre_{xy} , $Post_{xy}$, resp.).

A *marking* is a function that assigns to each discrete place a non-negative number of tokens, and to each continuous place a fluid volume. Therefore, $M : P_c \rightarrow \mathbb{R}_0^+$, $P_d \rightarrow \mathbb{N}$. The marking of place p_i is denoted M_i , while the value of the marking at time τ is denoted $M(\tau)$. The restriction of M to P_d and P_c are denoted with M^d and M^c , resp.

A *system* $\langle N, M(\tau_0) \rangle$ is an FOHPN N with an initial marking $M(\tau_0)$.

Net dynamics: The enabling of a discrete transition depends on the marking of all its input places, both discrete and continuous. More precisely, a discrete transition t is *enabled* at M if for all $p_i \in \bullet t$, $M_i \geq Pre(p_i, t)$, where $\bullet t$ denotes the preset of transition t . The *enabling degree* of t at M is equal to $enab(M, t) = \max\{k \in \mathbb{N} \mid M \geq k \cdot Pre(\cdot, t)\}$.

If t is *infinite-server semantics*, we associate to it a number of clocks that is equal to $enab(M, t)$. Each clock is initialized to a value that is equal to the time delay of t , if t is deterministic, or to a random value depending on the distribution function of t , if t is stochastic. If a discrete transition is *k-server semantics*, then the number of clocks that are associated to t is equal to $\min\{k, enab(M, t)\}$. The values of clocks associated to t decrease linearly with time, and t fires when the value of one of its clocks is null (if $\bar{k}$ clocks reach simultaneously a null value, then t fires $\bar{k}$ times). Note that here we are considering *enabling memory policy*, not *total memory policy*. This means that if a transition enabling degree is reduced by the firing of a different transition, then the disabled clocks have no memory of this in future enabling [8,1].

If a discrete transition t_j fires k times at time τ , then its firing at $M(\tau^-)$ yields a new marking $M(\tau)$ such that $M^c(\tau) = M^c(\tau^-) + \mathbf{C}_{cd}\boldsymbol{\sigma}$, and $M^d(\tau) = M^d(\tau^-) + \mathbf{C}_{dd}\boldsymbol{\sigma}$, where $\boldsymbol{\sigma} = k \cdot \mathbf{e}_j$ is the *firing count vector* associated to the firing of transition t_j k times.

Every continuous transition t_j is associated with an instantaneous firing speed (IFS) $v_j(\tau)$. For all τ it should be $V_j' \leq v_j(\tau) \leq V_j$, and the IFS of each continuous transition is piecewise constant between events.

A continuous transition is enabled only by the marking of its input discrete places. The marking of its input continuous places, however, is used to distinguish

between *strong* and *weakly enabled*: if all input continuous places of t_j have a not null marking, then t_j is called *strongly enabled*, else t_j is called *weakly enabled*¹.

We can write the equation which governs the evolution in time of the marking of a place $p_i \in P_c$ as $\dot{m}_i(\tau) = \sum_{t_j \in T_c} C(p_i, t_j) v_j(\tau)$ where $\mathbf{v}(\tau) = [v_1(\tau), \dots, v_{n_c}(\tau)]^T$ is the IFS vector at time τ .

The enabling state of a continuous transition t_j defines its admissible IFS v_j . If t_j is not enabled then $v_j = 0$. If t_j is strongly enabled, then it may fire with any firing speed $v_j \in [V'_j, V_j]$. If t_j is weakly enabled, then it may fire with any firing speed $v_j \in [V'_j, \bar{V}_j]$, where $\bar{V}_j \leq V_j$ since t_j cannot remove more fluid from any empty input continuous place $\bar{p}$ than the quantity entered in $\bar{p}$ by other transitions. Linear inequalities can be used to characterize the set of *all* admissible firing speed vectors $\mathcal{S}$. Each vector $\mathbf{v} \in \mathcal{S}$ represents a particular mode of operation of the system described by the net, and among all possible modes of operation, the system operator may choose the best, i.e., the one that maximize a given performance index $J(\mathbf{v})$ [5].

We say that a *macro-event* (ME) occurs when: (a) a discrete transition fires, thus changing the discrete marking and enabling/disabling a continuous transition; (b) a continuous place becomes empty, thus changing the enabling state of a continuous transition from strong to weak; (c) a continuous place, whose marking is increasing (decreasing), reaches a flow level that increases (decreases) the enabling degree of discrete transitions.

Let τ_k and τ_{k+1} be the occurrence times of two consecutive ME as defined above; we assume that within the interval of time $[\tau_k, \tau_{k+1})$, denoted as a *macro-period* (MP), the IFS vector is constant and we denote it $\mathbf{v}(\tau_k)$. Then, the continuous behavior of an FOHPN for $\tau \in [\tau_k, \tau_{k+1})$ is described by $M^c(\tau) = M^c(\tau_k) + \mathbf{C}_{cc}\mathbf{v}(\tau_k)(\tau - \tau_k)$, $M^d(\tau) = M^d(\tau_k)$.

Example 1. Consider the net system in Fig. 1(a). Place p_1 is a continuous place, while all other places are discrete. Continuous transitions t_1, t_2 have MFS $V_1 = 1$, $V_2 = 2$ and null mfs. Deterministic timed discrete transitions t_3, t_5 have timing delays 2 and 1.5, resp. Exponential stochastic discrete transitions t_4, t_6 have average firing rates are $\lambda_4 = 2$ and $\lambda_6 = 1.5$.

The continuous transitions represent two unreliable machines; parts produced by the first machine (t_1) are put in a buffer (p_1) before being processed by the second machine (t_2). The discrete subnet represents the failure model of the machines. When p_3 is marked, t_1 is enabled, i.e. the first machine is operational; when p_2 is marked, transition t_1 is not enabled, i.e. the first machine is down. A similar interpretation applies to the second machine.

Assume that we want to maximize the production rates of machines. In such a case, during any MP continuous transitions fire at their highest speed. This means that we want to maximize $J(\mathbf{v}) = v_1 + v_2$ under the constraints $v_1 \leq V_1$, $v_2 \leq V_2$, and — when p_1 is empty — $v_2 \leq v_1$.

¹ We are using an enabling policy for continuous transitions slightly different from the one proposed by David and Alla [1]. See [5] for a detailed discussion.

The resulting *evolution graph* and the time evolution of M_1 , v_1 and v_2 are shown in Fig. 1(b) and (c). During the first MP (of length 1) both continuous transitions are strongly enabled and fire at their MFS. After one time unit, p_1 gets empty, thus t_2 becomes weakly enabled fires at the same speed of t_1 . At time $\tau = 1.5$, transition t_5 fires, disabling t_2 . ■

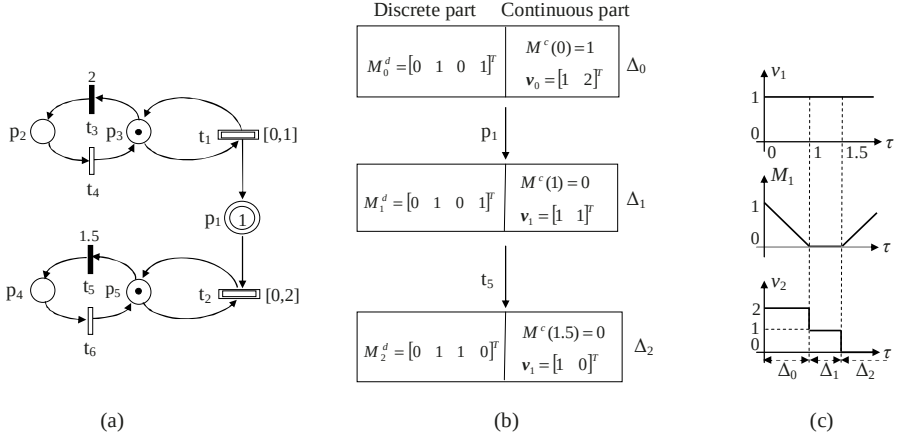


Fig. 1. (a) An FOHPN, (b) its evolution graph, and (c) its evolution in time

3 The HYPENS Tool

HYPENS has been developed in Matlab (Version 7.1). It is composed of 4 main files. The first two files, *make_HPN.m* and *enter_HPN.m* create the net to be simulated: the former requires input data from the workspace while the latter is a guided procedure.

The file *simulator_HPN.m* computes the timing evolution of the net that is summarized in an array of cells called *Evol*. Based on this array, the file *analysis_HPN.m* computes useful statistics and plots the simulation results.

Function make_HPN. $[Pre, Post, M0, vel, v, D, s, alpha] = \text{make\_HPN}(Pre_{cc}, Pre_{cd}, Pre_{dc}, Pre_{dd}, Post_{cc}, Post_{cd}, Post_{dd}, M0_c, M0_d, vel, v, D, s, alpha)$.

Input arguments

- Matrices $Pre_{cc}, Pre_{cd}, Pre_{dc}, Pre_{dd}, Post_{cc}, Post_{cd}, Post_{dd}$;
- The initial marking $M0_c$ and $M0_d$ of continuous/discrete places.
- Matrix $vel \in (\mathbb{R}_0^+)^{q_c \times 2}$ specifies, for each continuous transition, the mfs and the MFS.
- Vector $v \in \mathbb{N}^{1 \times q_d}$ specifies the timing structure of each discrete transition. The entries of this vector may take the following values: 1 - *deterministic*; 2 - *exponential distribution*; 3 - *uniform distribution*; 4 - *Poisson distribution*; 5 - *Rayleigh distribution*; 6 - *Weibull distribution*; etc.

– Matrix $D \in (\mathbb{R}_0^+)^{q_d \times 3}$ associates to each discrete transition a row vector of length 3. If the transition is deterministic, the first element of the row is equal to the time delay of transition. If the transition is stochastic, the elements of the row specify the parameters of the corresponding distribution function (up to three, given the available distribution functions).

– Vector $s \in \mathbb{N}^{1 \times q_d}$ keeps track of the number of servers associated to discrete transitions. The entries take any value in $\mathbb{N}$: 0 if the corresponding transition has *infinite servers*; $k > 0$ if the corresponding transition has k *servers*.

– Vector $alpha$ specifies the conflict resolution policy among discrete transitions.

- If $alpha \in \mathbb{N}^{1 \times q_d}$ two cases are possible. If all its entries are zero, conflict resolution is solved by priorities that depend on the indices of transitions (the smallest the index, the highest the priority). Otherwise, all its entries are greater than zero and specify the weight of the corresponding transition, i.e., if T_e is the set of enabled transitions, the probability of firing transition $t \in T_e$ is $\pi(t) = alpha(t) / (\sum_{t' \in T_e} alpha(t'))$.
- If $alpha \in \mathbb{N}^{2 \times q_d}$ the first row specifies the weights associated to transitions (as in the previous case) while the second row specifies the priorities associated to transitions. During simulation, when a conflict arises, priority are first considered; in the case of equal priority, weights are used to solve the conflict. See [1] for details.

Output arguments. (They are nothing else than input data, appropriately rewritten to be passed to function `simulator_HPN`).

– Matrices Pre and $Post$ are defined as:

$$Pre = \begin{bmatrix} Pre_{cc} & NaN & Pre_{cd} \\ NaN & NaN & NaN \\ Pre_{dc} & NaN & Pre_{dd} \end{bmatrix}, \quad Post = \begin{bmatrix} Post_{cc} & NaN & Post_{cd} \\ NaN & NaN & NaN \\ Post_{dc} & NaN & Post_{dd} \end{bmatrix}$$

where a row and a column of *NaN* (not a number) have been introduced to better visualize the continuous and/or discrete sub-matrices.

– The initial marking is denoted as $M0$ and is defined as a column vector.

– All other output data are identical to the input data.

Function `enter_HPN`: $[Pre, Post, M0, vel, v, D, s, alpha] = enter\_HPN$.

This function creates the net following a guided procedure. The parameters are identical to those defined for the previous function `make_HPN.m`.

Function `simulator_HPN`: $Evol = simulator\_HPN(Pre, Post, M, vel, v, D, s, alpha, time\_stop, simulation\_type, J)$.

Input arguments. They coincide with the output argument of the previous interface functions, plus three additional parameters.

– *time_stop* is equal to the time length of simulation.

– *simulation_type* $\in \{2, 1, 0\}$ specifies the simulation mode. Mode 0: no intermediate result is shown but only array *Evol* is created to be later analyzed

analysis_HPN. Modes 1 and 2 generate on screen the *evolution graph*: in the first case the simulation proceeds without interruptions until *time_stop* is reached; in the second case the simulation is carried out step-by-step.

– $J \in \mathbb{R}^{1 \times q_c}$ is a row vector that associates to each continuous transition a weight: $J \cdot v$ is the linear cost function that should be maximized at each MP to compute the IFS vector v . This optimization problem is solved using the subroutine *glpk mex.m* of Matlab.

Output arguments. The output is an array of cells called *Evol*, with the following entries (here K is the number of ME that occur during the simulation run).

- *Type* $\in \{1, 2, 3\}$: 1 (2, 3) if the net is continuous (discrete, hybrid).
- $M\_Evol \in (\mathbb{R}_0^+)^{n \times (K+1)}$ keeps track of the marking of the net during all the evolution: an n -dimensional column vector is associated to the initial time instant and to all the time instants in which a different ME occurs, each one representing the corresponding value of the marking at that time instant.
- $IFS\_Evol \in (\mathbb{R}_0^+)^{q_c \times (K+1)}$ keeps track of the IFS vectors during all the evolution. In particular, a q_c -dimensional column vector is associated to the initial configuration and to the end of each ME.
- *P_macro* and *Event_macro_Evol* are $(K + 1)$ -dimensional row vectors and keep track of the ME caused by continuous places. If the generic r -th ME is due to continuous place p_j , then the $(r+1)$ -th entry of *P_macro* is equal to j ; if it is due to the firing of a discrete transition, then the $(r+1)$ -th entry of *P_macro* is equal to *NaN*. The entries of *Event_macro_Evol* may take values in $\{0, 1, -1, NaN\}$: 0 means that the corresponding continuous place gets empty; 1 means that the continuous place enables a new discrete transition; -1 means that the continuous place disables a discrete transition. If the generic r -th ME is due to the firing of a discrete transition, then the $(r+1)$ -th entry of *Event_macro_Evol* is equal to *NaN* as well. The first entries of both *P_macro* and *Event_macro_Evol* are always equal to *NaN*.
- *firing_transition* is a $(K + 1)$ -dimensional row vector that keeps track of the discrete transitions that have fired during all the evolution. If the r -th ME is caused by the firing of discrete transition t_k , then the $(r + 1)$ -th entry of *firing_transition* is equal to k ; if it is caused by a continuous place, then the $(r + 1)$ -th entry of *firing_transition* is equal to 0. Note that the first entry of *firing_transition* is always equal to *NaN*.
- *timer_macro_event* is a $(K + 1)$ -dimensional row vector that keeps into memory the length of ME. The first entry is always equal to *NaN*.
- τ is equal to the total time of simulation.
- *Q_Evol* is a $((K + 1) \times q_d)$ -dimension array of cells, whose generic $(r + 1, j)$ -th entry specifies the clocks of transition t_j at the end of the r -th MP.
- $P_c\_P_d\_T_c\_T_d \in \mathbb{N}^4$ is a 4-dimensional row vector equal to $[n_c \ n_d \ q_c \ q_d]$.

Function analysis_HPN: $[P\_ave, P\_max, Pd\_ave\_t, IFS\_ave, Td\_ave, Pd\_freq, Md\_freq] = analysis\_HPN (Evol, static\_plot, graph, marking\_plot, Td\_firing\_plot,$

up_marking_plot, *Pd_prob_plot*, *Pd_ave_t_plot*, *IFS_plot*, *up_IFS_plot*, *IFS_ave_plot*, *Td_freq_plot*).

This function computes useful statistics and plots the results of the simulation run contained in *Evol*.

Input arguments. – *statistic_plot* $\in \{0, 1\}$: if 1 two histograms are created showing for each place, the maximum and the average marking during the simulation.

– *graph* $\in \{0, 1\}$: when set to 1 the evolution graph is printed on screen.

– *marking_plot*: is a vector used to plot the marking evolution of selected places in separate figures. As an example, if we set *marking_plot* = $[x \ y \ z]$, the marking evolution of places p_x , p_y and p_z is plotted. If *marking_plot* = $[-1]$, then the marking evolution of all places is plotted.

– *Td_firing_plot* $\in \{0, 1\}$: when set to 1 a graph is created showing the time instants at which discrete transitions have fired.

– *up_marking_plot* is a vector used to plot the marking evolution of selected places in a single figure. The syntax is the same as that of *marking_plot*.

– *Pd_prob_plot* $\in \{0, 1\}$: 1 means that as many plots as the number of discrete places will be visualized, each one representing the frequency of having a given number of tokens during the simulation run.

– *Pd_ave_t_plot* is a vector used to plot the average marking in discrete places with respect to time. A different figure is associated to each place, and the syntax to select places is the same as that of *marking_plot*.

– *IFS_plot* (resp., *up_IFS_plot*): is a vector used to plot the IFS of selected transitions in separate figures (resp., in a single figure). The syntax is the same as that of *marking_plot* and *up_marking_plot*.

– *IFS_ave_plot* $\in \{0, 1\}$: if 1 an histogram is created showing the average firing speed of continuous transitions.

– *Td_freq_plot* $\in \{0, 1\}$: if 1 an histogram is created showing the firing frequency of discrete transitions during the simulation run.

Output arguments

– *P_ave* (*P_max*) $\in \mathbb{R}^{1 \times n}$: each entry is equal to the average (maximum) marking of the corresponding place during the simulation run.

– *Pd_ave_t* $\in \mathbb{R}^{n_d \times K}$: column k specifies the average marking of discrete places from time $\tau_0 = 0$ to time τ_k when the k -th ME occurs.

– *IFS_ave* $\in \mathbb{R}^{1 \times q_c}$: each entry is equal to the average IFS of the corresponding continuous transition.

– *Td_ave* $\in \mathbb{R}^{1 \times q_d}$: each entry is equal to the average enabling time of the corresponding discrete transition.

– *Pd_freq* $\in \mathbb{R}^{n_d \times (z+1)}$ specifies for each discrete place the frequency of a given number of tokens during the simulation run. Here z is the maximum number of tokens in any discrete place during the simulation run.

– *Md_freq* $\in \mathbb{R}^{(n_d+1) \times \tilde{K}}$ where $\tilde{K}$ denotes the number of different discrete markings that are reached during the simulation run. Each column of *Md_freq* contains one of such markings and its corresponding frequency as a last entry.

4 A Numerical Example

Let us consider again the FOHPN system in Fig. 1(a) with performance index to be maximized $J = v_1 + v_2$ and a simulation time of 20 time units. The most significant results of the simulation run are shown in Fig. 2: the marking evolution of continuous place p_1 , of discrete places p_3 and p_5 , the IFS v_1 and v_2 , and the occurrence of ME. In particular, the bottom right figure specifies if the ME is due to a discrete transition (the index i of transition is shown in the y axis) or to a continuous place (the value of the y axis is equal to zero).

The main results of function analysis_HPN.m are:

$$\begin{aligned}
 P\_ave &= \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 \\ 0.127 & 0.600 & 0.400 & 0.475 & 0.525 \end{bmatrix}, & P\_max &= \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 \\ 1.321 & 1 & 1 & 1 & 1 \end{bmatrix}, \\
 & \begin{matrix} 0 & 1 \\ t_1 & t_2 \end{matrix} & IFS\_ave &= \begin{bmatrix} 0.400 & 0.450 \end{bmatrix}, \\
 Pd\_freq &= \begin{matrix} p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix} \begin{bmatrix} 0.400 & 0.600 \\ 0.600 & 0.400 \\ 0.525 & 0.475 \\ 0.475 & 0.525 \end{bmatrix}, & & & \\
 & \begin{matrix} t_3 & t_4 & t_5 & t_6 \end{matrix} & Td\_ave &= \begin{bmatrix} 0.200 & 0.150 & 0.350 & 0.300 \end{bmatrix}.
 \end{aligned}$$

The evolution graph created by *simulator_HPN*, and matrices Md_ave_t , Md_freq are not reported here, but can be downloaded from the web site [9].

In this paper we only present a very simple numerical example but we proved the efficiency of HYPENS via real dimensional cases examples in [10]. Here we considered both timed nets, modeling a family of queueing networks, and a hybrid net modeling a job shop system with finite capacity buffers and unreliable multi-class machines.

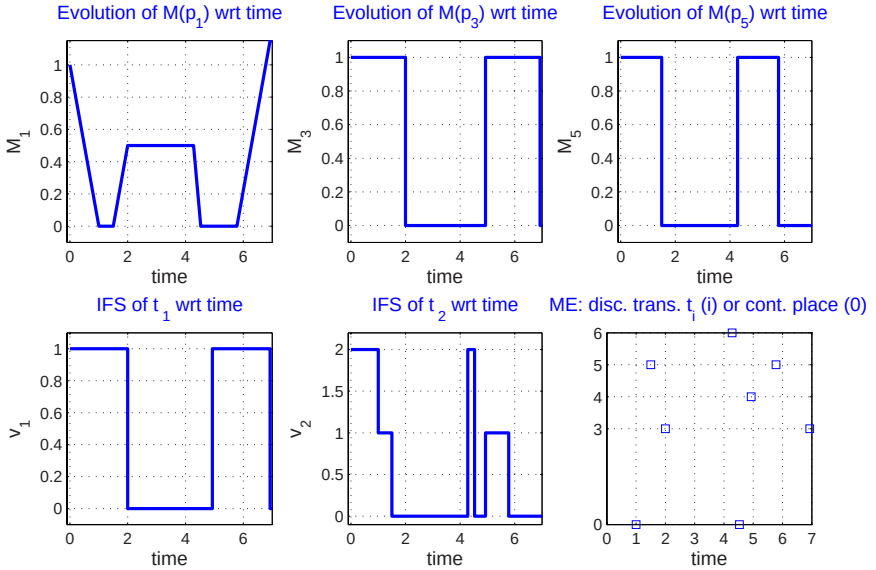


Fig. 2. Simulations carried out on the FOHPN system in Fig. 1(a) using HYPENS

5 Conclusions

We presented a Matlab tool for the simulation and analysis of timed discrete, continuous and hybrid PNs. Very general distribution functions can be considered, and the analysis of simulation may be efficiently carried out both graphically and numerically. The optimization of the net with respect to a given performance index may be easily carried out, and different conflict resolution policies may be considered. We plan to extend the tool adding the primitive "total memory policy".

References

1. David, R., Alla, H.: *Discrete, Continuous and Hybrid Petri Nets*. Springer, Heidelberg (2004)
2. Silva, M., Recalde, L.: On fluidification of Petri net models: from discrete to hybrid and continuous models. *Annual Reviews in Control* 28, 253–266 (2004)
3. Di Febbraro, A., Giua, A., Menga, G. (eds.): *Special issue on Hybrid Petri nets. Discrete Event Dynamic Systems* (2001)
4. Balduzzi, F., Menga, G., Giua, A.: First-order hybrid Petri nets: a model for optimization and control. *IEEE Trans. on Robotics and Automation* 16, 382–399 (2000)
5. Balduzzi, F., Giua, A., Seatzu, C.: Modelling and simulation of manufacturing systems with first-order hybrid Petri nets. *Int. J. of Production Research* 39, 255–282 (2001)
6. <http://sourceforge.net/projects/hisim>
7. <http://www.lag.ensieg.inpg.fr/sirphyco>
8. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: *Modelling with Generalized Stochastic Petri Nets*. Wiley, Chichester (1995)
9. <http://www.diee.unica.it/automatica/hypens>
10. Giua, A., Seatzu, C., Sessego, F.: Simulation and analysis of hybrid Petri nets using the Matlab tool HYPENS. In: *Proc. 2008 IEEE Int. Conf. on Systems, Man and Cybernetics*, Singapore (submitted, 2008)

Author Index

- Alonso, Gustavo 1
- Bergenthum, Robin 13, 388
- Best, Eike 33
- Billington, Jonathan 191
- Bonchi, Filippo 52
- Bouroulet, Roland 72
- Brogi, Antonio 52
- Cabac, Lawrence 399
- Carmona, J. 92
- Corfini, Sara 52
- Cortadella, J. 92
- Darondeau, Philippe 33, 112
- Desel, Jörg 388
- Devillers, Raymond 72
- Ding, Lay G. 132
- Dongen, B.F. van 368
- Dörges, Till 399
- Ehrenfeucht, A. 7
- Espensen, Kristian L. 152
- Fleischer, Paul 171
- Gadducci, Fabio 52
- Gallasch, Guy Edward 191
- Giua, Alessandro 419
- Gomes, Luís 409
- Hamez, Alexandre 211
- Hiraishi, Kunihiro 231
- Hurkens, C.A.J. 368
- Janicki, Ryszard 251
- Jantzen, Matthias 270
- Jifeng, He 8
- Khomenko, Victor 327
- Kishinevsky, M. 92
- Kjeldsen, Mads K. 152
- Klai, Kais 288
- Klaudel, Hanna 72
- Kondratyev, A. 92
- Kordon, Fabrice 211
- Koutny, Maciej 112
- Kristensen, Lars M. 152, 171
- Lavagno, L. 92
- Lê, Dai Tri Man 251
- Lime, Didier 307
- Lin, Huimin 9
- Liu, Lin 132
- Lorenz, Robert 13, 388
- Lourenco, João 409
- Magnin, Morgan 307
- Mauser, Sebastian 13, 388
- Meyer, Roland 327
- Oberheid, Hendrik 348
- Pelz, Elisabeth 72
- Petri, Carl Adam 12
- Pietkiewicz-Koutny, Marta 112
- Poitrenaud, Denis 288
- Pommereau, Franck 72
- Rölke, Heiko 399
- Roux, Olivier (H.) 307
- Rozenberg, G. 7
- Seatzu, Carla 419
- Serebrenik, A. 368
- Sessego, Fausto 419
- Söffker, Dirk 348
- Strazny, Tim 327
- Thierry-Mieg, Yann 211
- Werf, J.M.E.M. van der 368
- Yakovlev, Alex 92, 112
- Zetzsche, Georg 270