Lars M. Kristensen
Laure Petrucci (Eds.)

Applications and Theory of Petri Nets

LNCS 6709

32nd International Conference, PETRI NETS 2011
Newcastle, UK, June 2011
Proceedings

Springer

Lecture Notes in Computer Science 6709

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Lars M. Kristensen Laure Petrucci (Eds.)

Applications and Theory of Petri Nets

32nd International Conference, PETRI NETS 2011
Newcastle, UK, June 20-24, 2011
Proceedings

Springer

Volume Editors

Lars M. Kristensen
Bergen University College
Department of Computer Engineering
Nygaardsgaten 112, 5020 Bergen, Norway
E-mail: lmkr@hib.no

Laure Petrucci
Laboratoire d'Informatique de l'Université Paris Nord CNRS
UMR 7030, Institut Galilée
99, avenue Jean-Baptiste Clément, 93430, Villetaneuse, France
E-mail: petrucci@lipn.univ-paris13.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-21833-0 e-ISBN 978-3-642-21834-7
DOI 10.1007/978-3-642-21834-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011929347

CR Subject Classification (1998): F.1.1, D.2, F.3, H.4, D.1, F.4.1

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume constitutes the proceedings of the 32nd International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2011). 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. The satellite program of the conference comprised five workshops and three tutorials.

PETRI NETS 2011 was co-located with the 11th International Conference on Application of Concurrency to System Design (ACSD 2011). The two conferences shared five invited speakers. The PETRI NETS 2011 conference initially was to be organized by the Japan Advanced Institute of Science and Technology (JAIST) and hosted by Kanazawa University. Due to the earthquake and tsunami that struck Japan in March 2011, it was decided that both conferences be relocated to Newcastle upon Tyne, UK, and hosted by Newcastle University. A joint Japan–UK Organizing Committee was formed to ensure a successful relocation of the conference which was then held in Newcastle upon Tyne, UK, during June 20–24, 2011. We would like to express our deepest thanks to the joint Organizing Committee chaired by K. Hiraishi (Japan), M. Koutny (UK), and A. Yakovlev (UK), for all the time and effort invested in the local organization of the conference and its relocation.

This year the number of submitted papers amounted to 49, which included 40 full papers and 9 tool papers. The authors of the papers represented 20 different countries. We thank all the authors who submitted papers. Each paper was reviewed by at least three referees. For the first time the Programme Committee (PC) meeting took place electronically, using the EasyChair conference system for the paper selection process. The PC selected 17 papers: 13 regular papers and 4 tool papers for presentation. After the conference, some authors were invited to publish an extended version of their contribution in the *Fundamenta Informaticae* journal. We thank the PC members and other reviewers for their careful and timely evaluation of the submissions before the meeting, and the fruitful discussions during the electronic meeting. Finally, we are grateful to the invited speakers for their contribution: Brian Randell, Alessandro Giua, Monika Heiner, Walter Vogler, and Tomohiro Yoneda. The Springer LNCS team and the EasyChair system provided high-quality support in the preparation of this volume.

On behalf of the PETRI NETS community, we express our support and best wishes to our Japanese colleagues and friends in view of the tragic events that occurred in Japan in March 2011.

April 2011

Lars M. Kristensen
Laure Petrucci

Organization

Steering Committee

W. van der Aalst, The Netherlands
J. Billington, Australia
G. Ciardo, USA
J. Desel, Germany
S. Donatelli, Italy
S. Haddad, France
K. Hiraishi, Japan
K. Jensen, Denmark (Chair)
J. Kleijn, The Netherlands
M. Koutny, UK
C. Lin, China
W. Penczek, Poland
L. Pomello, Italy
W. Reisig, Germany
G. Rozenberg, The Netherlands
M. Silva, Spain
A. Valmari, Finland
A. Yakovlev, UK

Programme Committee

G. Balbo	University of Turin, Italy
M. Bednarczyk	Polish Academy of Sciences, Poland
J. Billington	University of South Australia, Australia
M. Bonsangue	Leiden University, The Netherlands
D. Buchs	University of Geneva, Switzerland
J. Carmona	Technical University of Catalonia, Spain
P. Chrzastowski-Wachtel	Warsaw University, Poland
G. Ciardo	University of California at Riverside, USA
J-M. Colom	University of Zaragoza, Spain
J. Desel	Katholische Universität Eichstätt-Ingolstadt, Germany
R. Devillers	Université Libre de Bruxelles, Belgium
J. Esparza	Technische Universität München, Germany
D. Fahland	Humboldt-Universität zu Berlin, Germany
Q-W. Ge	Yamaguchi University, Japan
A. Giua	University of Cagliari, Italy
L. Gomes	Universidade Nova de Lisboa, Portugal
S. Haddad	ENS Cachan, France
M. Heiner	Brandenburg University at Cottbus, Germany
K. Hiraishi	Japan Advanced Institute of Science and Technology, Japan
R. Janicki	McMaster University, Canada

E. Kindler	Danish Technical University, Denmark
L. Kristensen	Bergen University College (Co-chair), Norway
J. Lilius	Åbo Akademi University, Finland
C. Lin	Tsingnhua University, China
D. Moldt	University of Hamburg, Germany
M. Mukund	Chennai Mathematical Institute, India
W. Penczek	Polish Academy of Sciences, University of Podlasie, Poland
L Petrucci	Université Paris 13 (Co-chair), France
L. Pomello	Università di Milano-Bicocca, Italy
O-H. Roux	IRCCyN Nantes, France
N. Sidorova	Technical University of Eindhoven, The Netherlands
S. Taoka	Hiroshima University, Japan
V. Valero	University of Castilla-La Mancha, Spain
A. Valmari	Tampere University of Technology, Finland
A. Yakovlev	University of Newcastle, UK

General and Organizing Chairs

J. Atkinson, UK	K. Hiraishi, Japan
M. Koutny, UK	A. Yakovlev, UK

Workshop and Tutorial Chairs

W. van der Aalst, The Netherlands	J. Kleijn, The Netherlands

Tools Exhibition Chairs

V. Khomenko, UK	S. Yamane, Japan

Finance Chairs

K. Ogata, Japan	C. Smith, UK

Publicity Chairs

K. Kobayashi, Japan	J. Steggles, UK

Local Arrangements Chair

D. Carr, UK

Additional Reviewers

Arnold, Sonya
Barkaoui, Kamel
Barros, João Paulo
Bernardinello, Luca
Boucheneb, Hanifa
Cabac, Lawrence
Clariso, Robert
Costa, Aniko
Duflot, Marie
Duvigneau, Michael
Ferigato, Carlo
Fernandes, João Miguel
Franceschinis, Giuliana
Gallasch, Guy
Geeraerts, Gilles
Gribaudo, Marco
Haar, Stefan
Hansen, Henri
Hillah, Lom Messan
Hostettler, Steve
Khomenko, Victor
Koutny, Maciej
Lakos, Charles
Lasota, Sławomir
Liu, Fei
Liu, Lin
Lopez, Edmundo
Macià, Hermenegilda
Mangioni, Elisabetta
Meski, Artur
Miyamoto, Tosiyuki
Mlodzki, Marcin
Nakamura, Morikazu
Ochmanski, Edward
Ohta, Atsushi
Pawlowski, Wieslaw
Polrola, Agata
Pradat-Peyre, Jean-François
Quenum, José
Queudet, Audrey
Ribeiro, Oscar
Risoldi, Matteo
Rohr, Christian
Rosa-Velardo, Fernando
Schmitz, Sylvain
Schwarick, Martin
Sereno, Matteo
Solé, Marc
Stahl, Christian
Szreter, Maciej
Tiusanen, Mikko
Van Der Werf, Jan Martijn
Van Dongen, Boudewijn
Voorhoeve, Marc
Wagner, Thomas
Westergaard, Michael
Yamaguchi, Shingo
Zubkova, Nadezhda

Table of Contents

Invited Talks

Regular Papers

Tool Papers

Occurrence Nets Then and Now: The Path to Structured Occurrence Nets

Brian Randell

School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
United Kingdom
Brian.Randell@ncl.ac.uk

Abstract. This lecture, in honour of the late Carl Adam Petri, tells of my early interactions with him and summarizes a small sequence of research projects at Newcastle University from 1977 onwards that relate to occurrence nets, ending with a description of a planned new project on "structured occurrence nets". The areas of actual or planned application include deadlock avoidance, error recovery, atomicity, failure analysis, system synthesis and system verification.

Keywords: failures, errors, faults, dependability, judgement, occurrence nets, abstraction, formal analysis, deadlock avoidance, atomicity, failure analysis.

1 Introduction

I am greatly honoured to have been invited to give the Second Distinguished Carl Adam Petri Lecture. I say this because of the very limited extent, and often rather tangential nature, of my involvement with Carl Adam's and his followers' research. Indeed, I think it likely that most, probably all, of my audience here are far more familiar with this very large and valuable body of research than I am.

Nevertheless, despite my limited familiarity with your field, I have dared to devote this lecture to describing the modest series of net theory-related projects that I have been involved in over the years. Let me try to reassure you by saying that in each of these projects I worked in close co-operation with one or other of my colleagues. In each case this was someone who could readily provide the technical expertise in net theory that was needed to augment the very modest contributions that I was able to make to the research. Therefore, needless to say, I have relied heavily on the results of these colleagues' expertise in preparing this present lecture, a lecture that I humbly dedicate to the memory of a great computer scientist.

I'd like to start with a few words about my own personal memories of Carl Adam Petri. I believe I must have first learnt something of his work from Peter Lauer, who joined me at Newcastle University in 1972 from the IBM Vienna Laboratory. (I had reached Newcastle from the IBM Research Laboratory in Yorktown Heights in 1969, to take on the role of Director of Research in what was then termed the Computing Laboratory — now the School of Computing Science.) Starting in about 1975, Peter

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 1–16, 2011.

used Petri nets to provide a formal treatment [LAU1975] of the Path Expression concurrent programming notation. (This had been invented by Roy Campbell, then one of our PhD students, and Nico Habermann, of Carnegie-Mellon University [CAM1974].) In so doing Peter initiated a still-continuing and indeed flourishing line of research at Newcastle on concurrency theory, now led by my colleague Maciej Koutny.

Though my role in this concurrency research was that of an admiring bystander, I was intrigued and attracted by what I learnt of net theory from Peter and his colleagues. I assume that this is why, in 1976, I had no hesitation in inviting Carl Adam Petri to Newcastle for the first, and I fear only, time. This was to take part in our International Seminar on Teaching Computer Science. This series of annual seminars, to an audience mainly of senior computer science professors from across Europe, commenced in 1968 and continued for 32 years. The 1976 Seminar was on Distributed Computing System Design. Carl Adam Petri was one of six speakers. He gave three excellent lectures on the subject of "General Net Theory" [PET1977A] and on "Communication Disciplines" [PET1977B] — the written version of these lectures are, I now find, just the seventh and eighth of the thirty-eight publications listed, in date order, in the Prof. Dr. Carl Adam Petri Bibliography now provided online by TGI Hamburg.

I have just reread these lectures for the first time in many years and can readily understand why I was so impressed when I heard Carl Adam present them. But I cannot resist quoting his opening words:

> "Those of you who attended this conference last year may remember Anatol Holt's lecture 'Formal Methods in System Analysis'. My intention then, had been, this year to supplement his lecture by three hours of concentrated blackboard mathematics, because I felt that nothing needed adding to Holt's lectures in terms of words and figures. But I now think I ought to keep the mathematics to a minimum, both in order to give a general idea of the content of the theory, and to raise the entertainment value from negative to zero."

In fact the lectures provided a superb account of the basic concepts and the generality of the aims of net theory, illustrated by wonderfully simple yet subtle graphical examples — so I have taken great pleasure in making their text available online [PET1977A], [PET1977B].

To the best of my recollection the next occasion when I had the pleasure of meeting, and listening to lectures by, Carl Adam Petri was at the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979 [BRA1980]. To my pleased surprise I had been invited by the Course Director, Wilfried Brauer, to be one of the Course Co-Directors. Newcastle's more substantive contribution to the Course was however provided by Eike Best, then one of Peter Lauer's PhD students, who gave two of the lectures. (I will return to Eike Best shortly.) I can only assume that my involvement in this Course was what led to my being subsequently invited to be a member of the Progress Evaluation Committee that GMD established in 1981 to review the work of GMD's Carl Adam Petri Institute.

As a result I made a number of visits to Bonn in late 1981 and early 1982 to attend meetings of the Committee. It became clear to us all that the Committee had been set up by GMD's senior management in the expectation of, perhaps hope for, a negative evaluation. Indeed, it appeared they were seeking justification for closing Carl Adam

Petri's Institute down. We must have been a considerable disappointment to them. The Progress Evaluation Committee, though it made some minor constructive criticisms of the Institute, was in fact highly complimentary concerning the work of Carl Adam Petri and his colleagues. So this was a short episode in my career of which I am really quite proud, one that I'm pleased to have the opportunity of mentioning in this lecture.

But now let me turn to the net-related researches in which I have played a direct part myself. The earliest I wish to discuss (briefly) dates from 1977 — the latest, I'm pleased to say, is ongoing, and has been undertaken jointly with my colleague Maciej Koutny, and now also with Alex Yakovlev.

All of these researches have made use not of Petri Nets per se, but rather of Occurrence Nets. The first involved Phil Merlin, who spent some months with me at Newcastle in 1977 as a Senior Visiting Fellow (from IBM Research and the Technion, Haifa). Phil was a brilliant young researcher who was to die tragically at the age of just 32, just two years later. (The Technion honours his memory with the Dr. Philip M. Merlin Memorial Lecture and Prize Award.)

In his PhD Thesis, under Professor David Farber at UC Irvine, he did very interesting work on system recoverability, and so gained my and my colleagues' attention since the main, and still continuing, research interest at Newcastle was already system dependability. Specifically, in his PhD research Phil developed a technique for automating the conversion of a system design, expressed as a Petri Net, into an equivalent augmented Petri Net which would still work correctly in the presence of certain types of fault, such as arbitrary loss of single tokens [MER1974]. Subsequently, at IBM Research, he had worked on store and forward message passing — work that he had not been allowed to discuss in detail when he'd made a brief earlier visit to Newcastle, but which IBM had declassified by the time of his 1977 visit to us.

2 Deadlock Avoidance

Phil's time with us as a Senior Visiting Fellow coincided with a period when I and colleagues were actively involved in developing a large computer-controlled model railway. I'd seen the small model railway and the Scalextric car racing circuit that were provided for real time programming exercises at the University of Toronto, where I'd recently spent a very pleasant sabbatical. However it struck me that if a railway track was constructed out of a large set of separately powered sections of track, then a number of trains could be controlled individually. (This was long before the availability of digitally controlled model railways, embodying microprocessors in each engine that could be controlled separately by signals sent along the rails, but logically the effect was similar.)

Thus even just a single computer controlling the power supplies to the entire set of track sections, on which a set of "dumb" trains were running, could provide a challenging *concurrent* real time programming environment. In this environment each train was in effect an independent process, and one of the most obvious problems was how to control these "processes" so as to prevent them from crashing into each other!

Our train set had 32 separate sections, and ran half a dozen trains simultaneously. A large number of sensors each provided a simple indication when it was passed by a train.

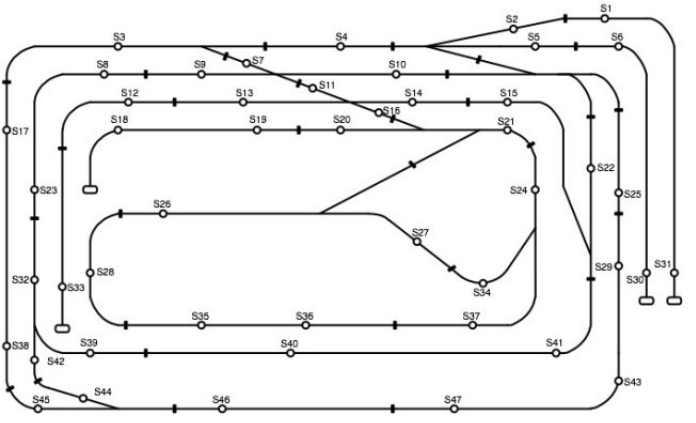


Fig. 1. The Model Railway Track Layout: bars and circles indicate the approximate positions of the section breaks and the train sensors ("stations")

The construction involved a number of colleagues in extensive hardware and software development. It was extremely successful, and led to a number of excellent individual and group student projects, and research investigations. (It was as far as I know the first such computer-controlled model railway, at least in any computer science department, anywhere — its story is told in [SNO1997].) I was closely involved in all this work, so when Phil started telling me about his work on deadlock prevention in store and forward message passing systems [MER1978A], I couldn't stop myself from trying to see how this work might relate to the train set.

Phil's work was all about buffer management — the problem of allocating buffers to messages at messaging nodes in sufficient numbers to allow continued progress. But our train set, which in effect controlled the movement of trains from track section to track section (actually from sensor to sensor), did not have anything equivalent to multiple buffers. This is because it was necessary to ensure that no section ever held more than a single train. (In effect it was a train system in which each "station" had just one "platform".) Thus a problem that we'd already started thinking about, of how to control a set of trains that were making simultaneous pre-planned journeys in various parts of the track layout, was a scheduling rather than an allocation problem.

Within a single discussion lasting just a few hours, Phil and I came up with a solution to the train journey deadlock avoidance problem [MER1978B]. During this discussion he told me about "occurrence nets", or "causal nets", a notation with which he was already very familiar, one that he traced back to papers by Anatol Holt [HOL1968] and by Carl Adam Petri [PET1976]. In fact we made only informal use of occurrence nets in this brief investigation, but the deadlock avoidance problem that we had identified and solved was christened the "Merlin-Randell problem" and taken up by some of our more theoretical colleagues at Newcastle and elsewhere.

In particular, it was the subject of Maciej Koutny's PhD Thesis at Warsaw Institute of Technology. (The thesis [KOU1984A], which gave the first formal treatment of the problem, is in Polish — the main results, though without proofs, are given in [KOU1984B].) Soon after completing his thesis Maciej joined Newcastle, and for a while continued

his study of the Merlin-Randell problem — in [KOU1985] he provided perhaps the best informal statement of the problem in the following terms:

> "There is a finite set of trains and a layout. The layout is represented by an undirected graph, the nodes of which represent places where trains can reside (stations), the arcs of which represent possible moves. Each station can hold only one train. Each train has a program to follow consisting of a directed path through the graph. The train can leave a station when the station it is immediately to travel to is empty. The problem is to find a synchronisation among train movements which allows parallel movements where possible and enables each journey to be completed."

This problem of finding such sets of synchronised train movements, which was in essence that of calculating a suitable occurrence net, remained in vogue for quite a while, much like the "Dining Philosophers Problem" before it; I took no part in this research, but I did spend quite a bit of time on various issues to do with controlling the train set, though my main concern remained that of system dependability.

3 Error Recovery

Newcastle's work on dependability, started in 1970, initially concerned the problem of tolerating residual design faults in simple sequential programs [**?**]. (I had formed the then unfashionable view that current work on proving programs correct would not suffice for large complex programs, and could perhaps be usefully complemented by work on software fault tolerance.) From this, we had soon moved on to considering the problems of faults in concurrent programs, and then in distributed computing systems.

Phil Merlin enthusiastically joined in on this research while he was with us, and worked with me on a particular (backward) error recovery problem, i.e. the task of restoring a distributed system to a previous state which it is hoped or believed preceded the occurrence of any existing errors. Our aim was to provide a formal treatment of this problem. The summary of our resulting paper [MER1978C] stated that:

> "The formalisation is based on the use of what we term "Occurrence Graphs" to represent the cause-effect relationships that exist between the events that occur when a system is operational, and to indicate existing possibilities for state restoration. A protocol is presented which could be used in each of the nodes in a distributed computing system in order to provide system recoverability in the face even of multiple faults."

We described occurrence graphs as similar to occurrence nets, differing mainly in that we viewed an occurrence graph as a dynamic structure that is "generated" as the system that it is modelling executes, and which contains certain additional information indicating which prior states have been archived and so are restorable. We described the state restoration problem in the following terms:

> "If an error is detected, a previous consistent state of the system should be restored at which it is possible to ignore those events and conditions which

> originally followed that state. By a "previous consistent state", we mean a state the system might have been in according to the cause-effect relationships between events and conditions, rather than one that had actually existed before. If the restored state is prior to the presumed set of events and conditions (i.e. the fault or faults) which caused the error, then the faults and their consequences can thus be effectively ignored."

We tackled the problems arising from concurrency in their full generality, so as to deal with the possibility of there being multiple concurrent faults, some even occurring during error recovery. The solution we produced was a decentralized recovery mechanism that we entitled the "chase protocol". We assumed that each node of a distributed system would hold a record of the part of the occurrence graph that related to its contribution to the overall system behaviour. Then each node would execute a protocol that had the effect of causing error recovery commands to "chase" through the dynamically-growing occurrence graph so as to overtake ongoing error propagations, and the nodes to co-operate in identifying a consistent set of restorable states.

The (informal) definition of our protocol, though fully detailed, is surprisingly brief. The problem of extending it to allow for "nested recovery", i.e. not just a single level, but rather for multiple levels, of error recovery turned out to be much more difficult; this problem was subsequently solved by Graham Wood, one of my PhD students [WOO1981].

All this work on chase protocols was an outgrowth of earlier work on concurrency in which we had identified the "domino effect" [RAN1975] — a situation in which a single error in one process could at worst cause each of a set of interacting processes to have to be backed up, possibly past all their archived prior states. Having identified this problem, most of our research on it had in fact been concerned with protective co-ordination schemes, akin to transactions, which would limit the domino effect through forcing a system to execute some of its (asynchronous) behaviour atomically, just in case errors might occur. Our first such co-ordination scheme, the "conversation", was described in [RAN1975].

A particularly noteworthy contribution to this work was that on *atomic actions* [LOM1977]. This was by another of our sabbatical visitors from IBM Research, David Lomet. (His atomic actions were both a programming language construct and a method of process structuring. They were in fact the forerunner of what was later known as the "transactional memory" model [LAR2008], which was first practically implemented by Herlihy and Moss [HER1993].)

4 Atomicity

A much more formal and extended analysis of the concept of atomicity and treatment of the problem of error recovery in distributed systems was provided in a paper that was a direct successor to the chase protocols and atomic actions papers [BES1981]. This was written largely by Eike Best, a PhD student of Peter Lauer. (I was very much a second author.) To quote this paper's summary:

> "We propose a generalisation of occurrence graphs as a formal model of computational structure. The model is used to define the "atomic occurrence" of

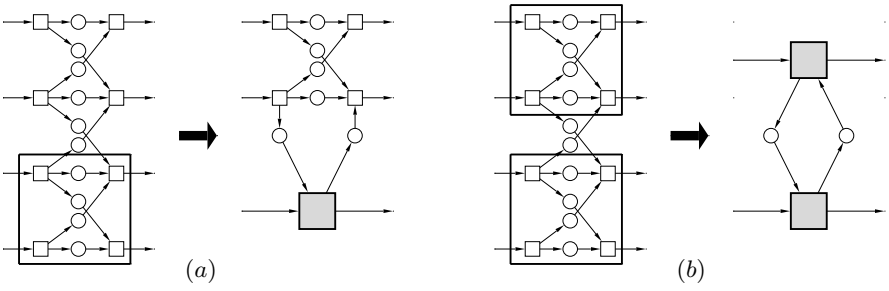


Fig. 2. (Invalid) Occurrence Graph Collapsing

> a program, to characterize "interference freeness" between programs, and to model error recovery in a decentralized system."

The generalization referred to involved the imposition of a "nested structure" of boxes onto a basic occurrence graph in order to produce what we termed a "structured occurrence graph". Such a graph is interpreted as indicating which sections of the original occurrence graph are intended to be viewed as being executed atomically, i.e. as ones that can be "collapsed" down to a single abstract event. Such collapsing is in fact a form of temporal abstraction, as shown in Figure 2.

This net collapsing operation can be rather tricky with occurrence nets that represent asynchronous activity, since there is a need to avoid it resulting in the introduction of any cycles into what is meant to be an acyclic directed graph. Figure 2 (b) shows the result of trying to view, simultaneously, two particular separate regions of an occurrence graph as being atomic. This in fact is not a valid result, since the collapsed graph contains a cycle.

This was perhaps the most interesting result in the above paper. There is much more that could be said about this paper, and about atomicity. I merely note that the subject is of major continuing interest to a number of different communities. Indeed, in recent years two very successful Dagstuhl Conferences have been held on atomicity, bringing together researchers from the fields of (i) database and transaction processing systems, (ii) fault tolerance and dependable systems, (iii) formal methods for system design and correctness reasoning, and (iv) hardware architecture and programming languages [JON2005]. However I instead wish to move on and discuss the most recent occurrence net-related work that I've been involved in. This in fact is the subject of the last, and main, topic of my lecture.

5 Structured Occurrence Nets

The work I want to describe now arose from my attempting to gain a deeper understanding of the relationships between the basic dependability concepts of failures, errors and faults when these occur in complex evolving asynchronous systems. The typical account of these concepts is as follows: A *failure* occurs when an *error* "passes through" the system-user interface and affects the service delivered by the system — a system of

course being composed of components which are themselves systems. This failure may be significant, and thus constitute a *fault*, to the enclosing system. Thus the manifestation of failures, faults and errors follows a *fundamental chain*:

. . . → failure → fault → error → failure → fault →. . .

i.e.

. . . → event → cause → state → event → cause → . . .

This fault-error-failure chain can flow from one system to: (i) another system that it is interacting with, (ii) the system of which it is part, and (iii) a system that it creates, modifies or sustains.

Typically, a failure will be judged to be due to multiple co-incident faults, e.g. the activity of a hacker exploiting a bug left by a programmer. Identifying failures (and hence errors and faults), even understanding the concepts, is difficult when:

- there can be uncertainties about system boundaries.
- the very complexity of the systems (and of any specifications) is a major difficulty.
- the determination of possible causes or consequences of failure can be a very subtle, and iterative, process.
- any provisions for preventing faults from causing failures may themselves be fallible.

Attempting to enumerate a system's possible failures beforehand is normally impracticable. What constitutes correct (failure-free) functioning might be implied by a system specification - assuming that this exists, and is complete, accurate and agreed. But the specification is often part of the problem!

The *environment* of a system is the wider system that it affects (by its correct functioning, and by its failures), and is affected by. In principle a third system, a *judgemental system*, is involved in determining whether any particular activity (or inactivity) of a system in a given environment constitutes or would constitute — from its viewpoint — a failure. Note that such a judgemental system might itself be fallible. (This possibility is allowed for in the legal system, hence the concept of a hierarchy of crown courts, appeal courts, supreme courts, etc., in the British legal system.)

Though I was moderately satisfied by the informal English-language accounts of all these concepts that have been developed and refined over many years in the IEEE and IFIP communities [AVI2004], I felt it desirable to find a more rigorous way of defining them.

5.1 Behavioural Abstraction

I started drawing sets of little example occurrence nets, trying to find a good way of illustrating failure/fault/error chains in complex evolving situations, and suddenly realised — to my great delight — that two concepts that I had been treating as logically distinct, namely system and state, are not separate, but just a question of abstraction, so that (different related) occurrence nets can represent both systems and their states using the same symbol — a *place*.

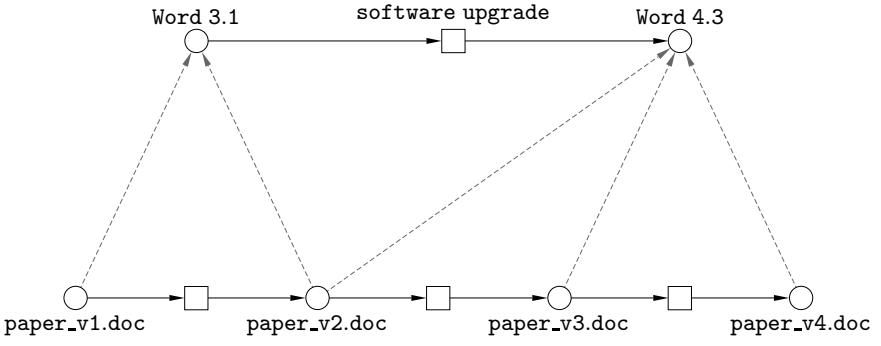


Fig. 3. An evolving software system

Figure 3 illustrates this form of abstraction, which I termed *behavioural abstraction*, by showing two occurrence nets, one portraying the changing state of a Microsoft WORD system as it undergoes modification, the other the sequence of updates made to a WORD document, the first set of which are made using WORD 3.1, with the rest of the document updates then being made using WORD 4.3.

In this figure dotted lines indicate how states in one occurrence net are related to sets of states in the other occurrence net. (Note that it is not possible to portray this entire evolving situation in a single occurrence net.)

I went on to explore (graphically, and very informally) a few other potentially useful types of relationships between occurrence nets, and started using the term *structured occurrence net* (SON) for a set of occurrence nets associated together by means of any of these various types of relationship. (I had forgotten until preparing this lecture that the term "structured occurrence graph" had already been used in [BES1981] for an occurrence graph on which a "nested structure" of atomicity boxes had been imposed. In fact the notion of a SON can be viewed as a major extension and generalization of such structured occurrence graphs.)

At this stage my motivation changed from trying merely to improve my understanding of the fault-error-failure chain concept. I became intrigued by the possible practical utility of SONs for supporting failure diagnosis, e.g. as the basis of a semi-automated means for organizing and analyzing the sort of large body of evidence that a court of inquiry has to deal with after a major accident, or the police are faced with during a complex criminal investigation.

Luckily Maciej Koutny became interested in my fumbling attempts at understanding and explaining what I was proposing. Indeed he took the lead by providing formal definitions of the growing number of relations that we identified as fundamental to SONs, and formulating and proving well-formedness theorems about them [RAN2007], [KOU2009]. (These theorems mainly concerned the need to maintain acyclicity, so that causality is not violated.)

We also defined some somewhat formal graphical conventions for the various types of SON — subsequent figures using these conventions are all taken from [KOU2009], as is much of the accompanying text. Dashed boxes are used to delineate occurrence nets. A box labelled "B-SON" bounds the set of occurrence nets that are related by means of

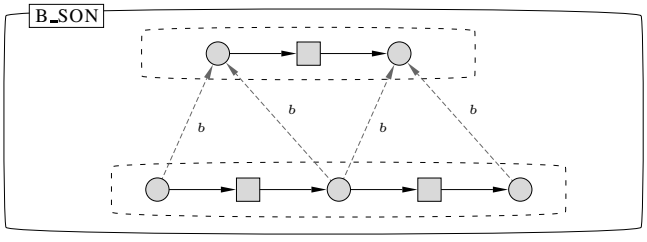


Fig. 4. Behavioural abstraction

behavioural abstraction, the actual relations between the elements of the two occurrence nets being shown by dashed arrows. Thus Figure 4 shows a behavioural abstraction that represents the sort of evolving software system that was portrayed informally in Figure 3.

Other relations, and hence means of structuring, that we defined included ones for communication, temporal abstraction, and spatial abstraction, each of which is briefly described below.

5.2 Communication

The *communication relation* is used for situations in which separate occurrence nets, portraying the activity of distinct systems, proceed concurrently and (occasionally) communicate with each other. See, for example, Figure 5(b), in which thick dashed arcs are used to represent communications so as to distinguish them from the interactions represented in conventional occurrence nets by causal arcs. Note that another distinction is that interactions *within* a conventional occurrence net link conditions to events and events to conditions, whereas communications link events — of *separate* occurrence nets — directly. (The C-SON label identifies the SON containing these separate occurrence nets as being one that has been defined using communication relations. Conditions and events of different systems are identified by shading them differently.)

In practice, when structuring a complex occurrence net into a set of simpler communicating occurrence nets (or building one from a set of component communicating occurrence nets), it is sometimes necessary to use synchronous communications — I return to this issue in the next section.

Hence, as shown in Figure 6, we allowed for the use of two types of communication: thick dashed directed arcs indicate, for example, that an event in one occurrence net is a causal predecessor of an event in another occurrence net (i.e., information flow between the two systems was unidirectional), whereas undirected such arcs indicate that the two events have been executed synchronously (i.e., information flow was bidirectional).

In practice, interactions and communications of all the kinds described above can occur in the same overall structured occurrence net provided that a simple acyclicity constraint — similar to that used for ordinary occurrence nets — is satisfied. (The utility of this and several of the other types of relation is not that they extend the types of system activity that can be portrayed but that they assist the use of abstraction and structuring in order to cope with complex situations. Moreover, the communications links may also be used to hide the details of the means by which occurrence nets interact, should this not be of interest.)

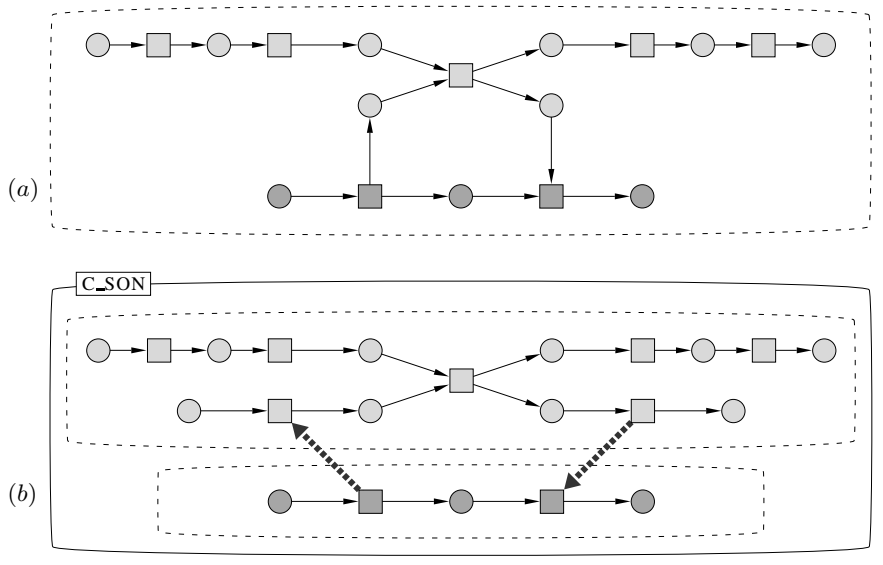


Fig. 5. Part (a) shows the activity of two systems in a single occurrence net; (b) shows an equivalent SON, in which the activities of these two systems are shown in separate (communicating) occurrence nets

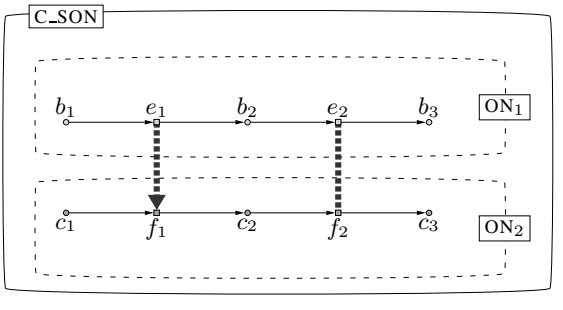


Fig. 6. Synchronous and Asynchronous Communication

5.3 Temporal Abstraction

When one "abbreviates" parts of an occurrence net one is in effect using *temporal abstraction* to define atomic actions, i.e., actions that appear to be instantaneous to their environment. These rules are best illustrated by an alternative representation for an occurrence net together with its abbreviations, namely a structured occurrence net in which each abbreviated section (or *atomic activity*) of the net is shown surrounded by an enclosing *event box*. Figure 7(b) shows this alternative representation of Figure 7(a), the top part of which can readily be recreated by "collapsing" Figure 7(b)'s occurrence net, i.e., by replacing the enclosed sections by simple event symbols, as shown in Figure 7(c).

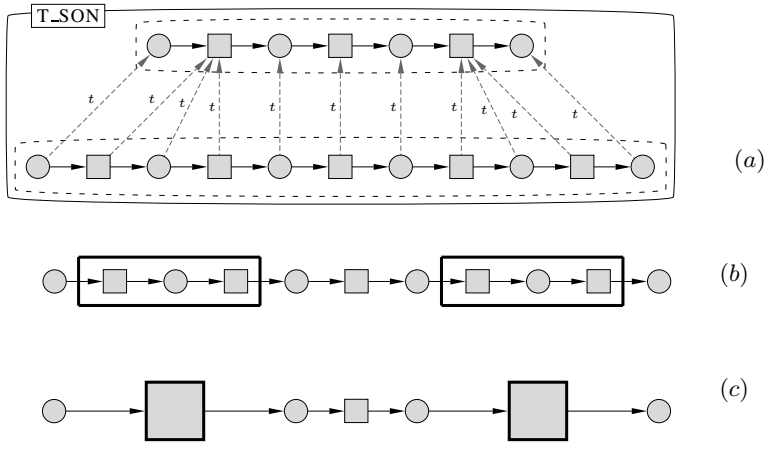


Fig. 7. Temporal abstraction

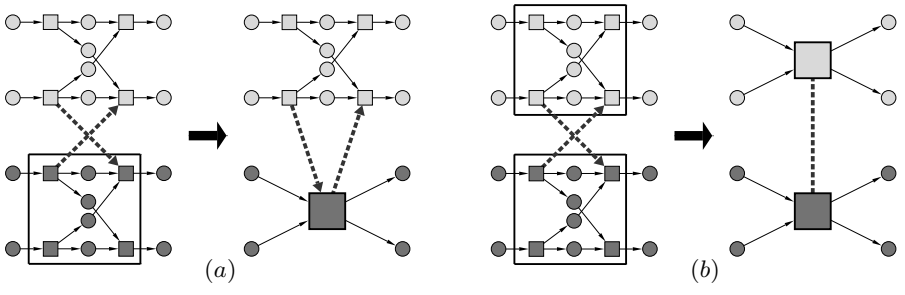


Fig. 8. Two valid collapsings that give rise to asynchronous (in (a)) and synchronous (in (b)) communication between abstract events

This net collapsing operation is much trickier with occurrence nets that represent asynchronous activity since there is a need to avoid introducing cycles into what is meant to be an acyclic directed graph — the problem that we had found much earlier in [BES1981] with structured occurrence graphs.

Hence the need, on occasion, to use *synchronous system* interactions, as shown in Figure 8, section (b) of which, in contrast to that of Figure 2, is acyclic and hence shows a *valid* collapsing.

5.4 Spatial Abstraction

What we called *spatial abstraction* is based on the relation "contains/is component of". Figure 9 shows the behaviour of a system and of the three systems of which it is composed, and how its behaviour is related to that of these components. (This figure does not represent the matter of *how*, or indeed whether, the component systems are enabled to communicate, i.e., what design is used, or what connectors are involved.

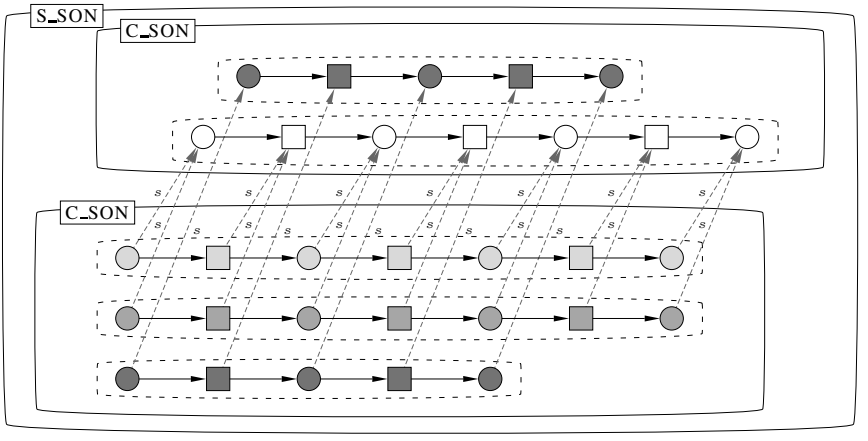


Fig. 9. Spatial abstraction. The 'spatially-abstracts' relation is indicated by s-labelled edges

Above I have presented composition and abbreviation, i.e., spatial and temporal abstraction, as though they are quite separate — in practice, it is likely that useful abstractions will result from successive applications of both spatial and temporal abstractions. The other formal relations that we defined for SONs include *information retention*, and *judgement*. Information retention is of relevance to the implementation of system recoverability and the gathering of evidence for later failure analysis; judgement is relevant to either such inline activities as failure detector circuits, or the retrospective activities of, for example, a court of enquiry. Preliminary studies have also been made of the problems of representing incomplete, contradictory and uncertain failure evidence [RAN2009]. However space constraints preclude the provision of further details.

5.5 Potential Uses of SONs

The various types of abstractions we defined for SONs are all ones that we believe could facilitate the task of understanding complex evolving systems and their failures, and of analyzing the cause(s) of such failures — given suitable tool support. The abstractions are likely, in most cases, to be a natural consequence of the way the systems have been conceived and perceived. Thus they can be viewed as providing a means of naturally structuring what would otherwise be an impossibly large and complex occurrence net.

Alternatively, they can be viewed as a way of reducing the combinatorial complexity of the information accumulated and the analyses performed in following fault-error-failure chains after the fact, e.g., in a safety inquiry. However we now view failure analysis as only one of the potential uses of SONs. Tools for system verification (e.g. [MCM1995], [KHO2007] and [ESP2008]) and for system synthesis (e.g. [BIL1996] and [KHO2006]) that currently embody the use of conventional occurrence nets could, we believe, gain significantly by exploiting the structuring possibilities provided by SONs. Specifically, SONs could enable such tools (i) to reduce their storage and computational resource requirements, or (ii) to cope with the representation and manipulation of more complex system behaviours than is currently practicable. (One can draw a direct

analogy to the way in which the structuring of large programs is exploited in many software engineering tools.)

Maciej Koutny, Alex Yakovlev and I are therefore now actively planning a research project aimed at designing and implementing a software tool for handling SONs and supporting their manipulation, visualisation and semantical analysis. This would embody the theoretical results detailed in [KOU2009] and be based on the WORKCRAFT platform [POL2009]. A further aim of the project would be to use the resulting tool in undertaking some initial investigations into the likely practical utility of SONs for failure analysis, and for system synthesis and verification.

6 Concluding Remarks

The small sequence of occurrence net-related projects that I have described here has, I like to think, a pleasing logic and coherence when viewed retrospectively — however this cannot be claimed to be a result of detailed planning. Rather it is, I believe, just a happy and perhaps natural consequence of the fact that they were undertaken in an environment of a slowly evolving community of enthusiastic and cooperative colleagues, sharing some fundamental long term common interests and attitudes. In particular I have in mind a deep concern for system dependability in its most general forms, and a delight in searching for simple solutions and (preferably recursive) system structuring principles. I hope therefore that this lecture will be taken as an appropriate, albeit I fear an inadequate, tribute to the work and memory of the late Carl Adam Petri.

Acknowledgements

As I indicated earlier, the work I have described here has been carried out in close collaboration with various colleagues, all of whose expertise in net theory greatly exceeds my own. It is a pleasure to acknowledge the debt that I owe to them all, and a necessity to indicate that any and all inadequacies in this account are mine alone.

References

[AVI2004] Avizienis, A., Laprie, J.C., et al.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1, 11–33 (2004)

[BES1981] Best, E., Randell, B.: A Formal Model of Atomicity in Asynchronous Systems. Acta Informatica 16, 93–124 (1981)

[BIL1996] Bilinski, K., Dagless, E.L., et al.: Behavioral Synthesis of Complex Parallel Controllers. In: Proc. of 9th International Conference on VLSI Design, pp. 186–191. IEEE, Los Alamitos (1996)

[BRA1980] Brauer, W. (ed.): Net Theory and Applications. Proceedings of the Advanced Course on General Net Theory of Processes and Systems. LNCS, vol. 340. Springer, Heidelberg (1980)

[CAM1974] Campbell, R., Habermann, A.N.: The Specification of Process Synchronization by Path Expressions. In: Proc. of Symposium on Operating Systems, pp. 89–102 (1981)

[ESP2008] Esparza, J., Heljanko, K.: Unfoldings: a Partial-order Approach to Model Checking. Springer, Heidelberg (2008)

[HER1993] Herlihy, M., Moss, J.E.B.M.: Transactional Memory: Architectural Support for Lock-Free Data Structures. In: Proc. of 20th Annual International Symposium on Computer Architecture, pp. 289–300 (1993)

[HOL1968] Holt, A.W., Shapiro, R.M., et al.: Information System Theory Project (Appl. Data Research [1] ADR 6606). RADC-TR-68-305, US Air Force, Rome Air Development Center (1968)

[HOR1974] Horning, J.J., Lauer, H.C., et al.: A Program Structure for Error Detection and Recovery. In: Gelenbe, E., Kaiser, C. (eds.) Operating Systems. LNCS, vol. 16, pp. 171–187. Springer, Heidelberg (1974)

[JON2005] Jones, C., Lomet, D.B., et al.: The Atomic Manifesto: A Story in Four Quarks. SIGMOD Record 34, 63–69 (2005)

[KHO2006] Khomenko, V., Koutny, M., et al.: Logic Synthesis for Asynchronous Circuits Based on STG Unfoldings and Incremental SAT. Fundamenta Informaticae 70, 49–73 (2006)

[KHO2007] Khomenko, V., Koutny, M.: Verification of Bounded Petri Nets Using Integer Programming. Formal Methods in System Design 30, 143–176 (2007)

[KOU1984A] Koutny, M.: O Problemie Pociagów Merlina-Randella (in Polish). PhD Thesis, Department of Mathematics, Warsaw University of Technology, Warsaw, Poland (1984)

[KOU1984B] Koutny, M.: On the Merlin-Randell Problem of Train Journeys. In: Paul, M., Robinet, B. (eds.) Programming 1984. LNCS, vol. 167, pp. 179–190. Springer, Heidelberg (1984)

[KOU1985] Koutny, M.: Train Journeys in the Merlin-Randell Problem. TR-205, Computing Laboratory, Newcastle University (1985)

[KOU2009] Koutny, M., Randell, B.: Structured Occurrence Nets: A formalism for Aiding System Failure Prevention and Analysis Techniques. Fundamenta Informaticae 97, 41–91 (2009)

[LAR2008] Larus, J., Kozyrakis, C.: Transactional Memory. Communications of the ACM 51, 80–88 (2008)

[LAU1975] Lauer, P.E., Campbell, R.H.: A Description of Path Expressions by Petri Nets. In: Proc. of 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 95–105. ACM, New York (1975)

[LOM1977] Lomet, D.B.: Process Structuring, Synchronization, and Recovery Using Atomic Actions. ACM SIGPLAN Notices 12, 128–137 (1977)

[MCM1995] McMillan, K.L.: A Technique of State Space Search Based on Unfolding. Formal Methods in System Design 6, 45–65 (1995)

[MER1974] Merlin, P.M.: A Study of the Recoverability of Computing Systems. PhD Thesis, University of California, Irvine (1974)

[MER1978A] Merlin, P.M., Schweitzer, P.J.: Deadlock Avoidance in Store-And-Forward Networks. In: Jerusalem Conference on Information Technology, pp. 577–581 (1978)

[MER1978B] Merlin, P.M., Randell, B.: Notes on Deadlock Avoidance on the Train Set. MRM/144, Computing Laboratory, Newcastle University (1978), `http://homepages.cs.ncl.ac.uk/brian.randell/MRM144.pdf`

[MER1978C] Merlin, P.M., Randell, B.: State Restoration in Distributed Systems. In: Proc. of 8th International Symposium on Fault-Tolerant Computing, pp. 129–134. IEEE Computer Society Press, Los Alamitos (1978)

[PET1976] Petri, C.A.: Nicht-sequentielle Prozesse. ISF-Bericht ISF-76-6, Gesellschaft für Mathematik und Datenverarbeitung Bonn (1976)

[PET1977A] Petri, C.A.: General Net Theory. In: Proc. of 1976 Joint IBM/Univ. of Newcastle upon Tyne Seminar on Computing System Design, Newcastle upon Tyne, Computing Laboratory , pp. 131–169 (1977) , http://homepages.cs.ncl.ac.uk/brian.randell/GeneralNetTheory.pdf

[PET1977B] Petri, C.A.: Communication Disciplines. In: Proc. of 1976 Joint IBM/Univ. of Newcastle upon Tyne Seminar on Computing System Design, Newcastle upon Tyne, Computing Laboratory, pp. 171–183 (1977), http://homepages.cs.ncl.ac.uk/brian.randell/CommunicationDisciplines.pdf

[POL2009] Poliakov, I., Khomenko, V., et al.: Workcraft — A Framework for Interpreted Graph Models. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 333–342. Springer, Heidelberg (2009)

[RAN1975] Randell, B.: System Structure for Software Fault Tolerance. IEEE Trans. on Software Engineering SE-1, 220–232 (1975)

[RAN2007] Randell, B., Koutny, M.: Failures: Their definition, modelling and analysis. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 260–274. Springer, Heidelberg (2007)

[RAN2009] Randell, B., Koutny, M.: Structured Occurrence Nets: Incomplete, Contradictory and Uncertain Failure Evidence. TR-1170, School of Computing Science, Newcastle University (2009)

[SNO1997] Snow, C.R.: Train Spotting. Newcastle upon Tyne, Computing Laboratory, Newcastle University (1997), http://history.cs.ncl.ac.uk/anniversaries/40th/webbook/trainset/index.html

[WOO1981] Wood, W.G.: A Decentralised Recovery Control Protocol. In: Proc. of 11th International Symposium on Fault-Tolerant Computing, pp. 159–164. IEEE Computer Society Press, Los Alamitos (1981)

How Might Petri Nets Enhance Your Systems Biology Toolkit

Monika Heiner* and David Gilbert

School of Information Systems, Computing and Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH, UK,
{monika.heiner,david.gilbert}@brunel.ac.uk

Abstract. "How might Petri nets enhance my Systems Biology toolkit?" – this is one of the questions that we get on a regular basis, which motivated us to write an answer in the form of this paper.

We discuss the extent to which the Petri net approach can be used as an umbrella formalism to support the process of BioModel Engineering. This includes the facilitation of an active and productive interaction between biomodellers and bioscientists during the construction and analysis of dynamic models of biological systems. These models play a crucial role in both Systems Biology, where they can be explanatory and predictive, and synthetic biology, where they are effectively design templates. In this paper we give an overview of the tools and techniques which have been shown to be useful so far, and describe some of the current open challenges.

Keywords: BioModel Engineering; Systems Biology; synthetic biology biomolecular networks; qualitative, stochastic and continuous Petri nets; model checking.

1 Motivation

Biology is increasingly becoming an informational science. This revolution has been driven by technological advances which have supported the development of studies at many levels of intra- and intercellular activity. These advances have facilitated the analysis of how the components of a biological system interact functionally - namely the field of Systems Biology [39]. In general this analysis is quantitative and over time [1], which can basically be done in a stochastic or continuous fashion.

At the heart of this field lies the construction of models of biological systems, see Figure 1. These models are used for analysis which should ideally be both *explanatory* of biological mechanisms and *predictive* of the behaviour of the system when it is perturbed by, e.g., mutations, chemical interventions or changes in the environment. Furthermore, models can be used to help make genetic engineering easier and more reliable, serving as design templates for novel synthetic biological systems – an emerging discipline known as synthetic biology [19,26]. Central

* *On sabbatical leave from Brandenburg University of Technology.*

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 17–37, 2011.

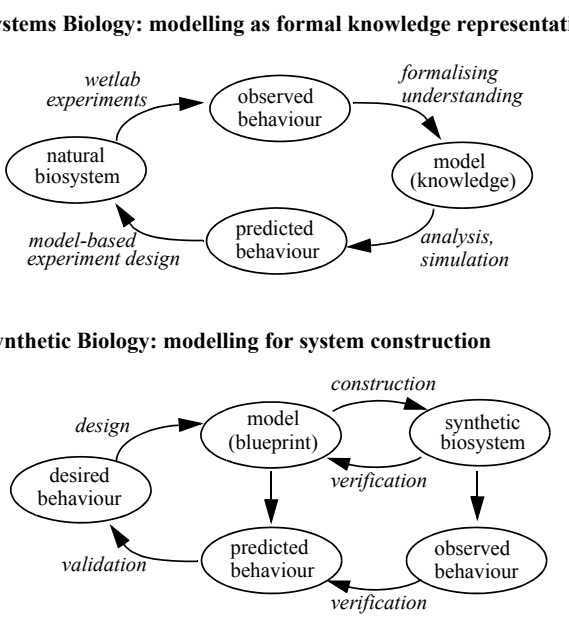


Fig. 1. The role of formal models in Systems and Synthetic Biology

to both Systems and Synthetic Biology is BioModel Engineering (BME) which is the science of designing, constructing and analyzing computational models of biological systems [8].

In this paper we discuss the Petri net approach to BioModel Engineering. We show how Petri nets are highly suited to the support of both Systems and Synthetic Biology. Not only are they immediately attractive to life scientists in terms of their graphical representation, thus facilitating communication with the modeller, but they permit qualitative as well as quantitative (stochastic, continuous, hybrid) descriptions by a family of related models and come with a very powerful analytical toolkit [29].

The structure of this paper is as follows: in Section 2 we discuss representational issues, and in Section 3 we recall our overall conceptual framework, followed by an overview of Petri net contributions in Section 4. We discuss model checking and its use in BioModel Engineering in Section 5, and summarize our toolkit in Section 6. Then we discuss the outlook and challenges for the use of Petri nets in this area in Section 7, and give an overall summary in Section 8.

This paper is deliberately kept informal. We assume a basic understanding of Petri nets; a gentle introduction within the given context of BioModel Engineering can be found in [48, 28], for related formal definitions see [29].

2 Representation Style – Just a Matter of Taste?

We consider biochemical *reaction networks* (RN), consisting of a finite set of

- *reactions* (biochemical reactions, complexation/decomplexation, phosphorylation/dephosphorylation, conformational change, transport steps, ..., any type of biochemical interactions, which are considered to be atomic on the chosen abstraction level), converting or transporting
- *species* (biochemical compounds, proteins, protein conformations, complexes, ..., any species, which are considered to be atomic on the chosen abstraction level).

Reactions may carry elementary kinetic information, *the reaction rates*, if known. A typical pattern of elementary reaction rates is the *mass/action kinetics*, where the state-dependent rate function v_i of the reaction r_i is assumed to be given by the product of the current values of the species involved and some further constants k_i (usually called reaction parameters), summing up environmental conditions such as temperature, pressure, ...; for examples see Figure 2, left and Figure 3, right.

Thus, the notion of a reaction network covers both the qualitative and quantitative modelling paradigms. A reaction network can be specified as:

(1) *list* (or set) of all individual (stoichiometric) reactions, in a reaction-centric or species-centric style,

or a graph describing all individual reactions, which could be either

(2) *hypergraph*, i.e., a graph where an arc can connect any number of nodes,
(3) *bipartite graph*, i.e., a graph with two types of nodes, whereby arcs never connect nodes of the same type; e.g. a Petri net.

Remark: In the Petri net examples we adopt the usual interpretation and represent reactions (the active system components) by transitions, and species (the passive system components) by places.

These three different styles of representations (1)–(3) can be converted into each other without loss of information. There are two well-known representations, which can be uniquely derived from any of (1)–(3), but generally not vice versa:

(4) *incidence matrix* (in Systems and Synthetic Biology better known as stoichiometric matrix),
(5) *system of ordinary differential equations* (ODEs).

Both derived styles of representation lose structural information if there are catalysts involved or species, which are both the subtrate as well as the product of one and the same reaction. It is an easy exercise to imagine two reaction networks generating the same incidence matrix or ODEs, but differing in their discrete behaviour. We start from the introductory example in [61], see Fig. 2,

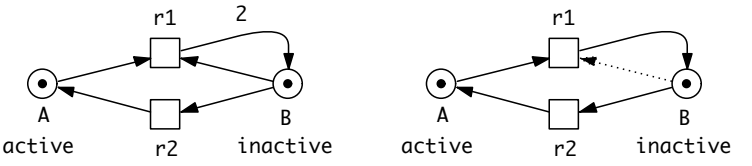


Fig. 2. Two reaction networks generating the same incidence matrix and the same ODEs. With the kinetic rates $v_1 = v(r_1) = k_1 \cdot A \cdot B, v_2 = v(r_2) = k_2 \cdot B$, we get the two equations $dA/dt = v_2 - v_1$, $dB/dt = v_1 - v_2$.

left. The net on the right uses a modifier arc (dotted line), which is a standard feature of the Systems Biology Markup Language (SBML) [41] to indicate that the rate of r_1 dependents on B. For more examples see [63].

Consequently, the structural interpretation of ODEs models, i.e. converting (5) into any of (1)–(4), calls for special attention. The problematic cases are reactions with complex kinetics. Some tools popular in Systems Biology, e.g., COPASI [40], provide predefined functions representing whole building blocks, i.e. subnets in our reaction networks. See Fig. 3 for an example, which has been taken from the reaction network reported in [9]. The network on the left has the structure as suggested by the schematic representation in [9] and the list of reactions in the model's SBML format created by COPASI. The network on the right shows the correct structure, which is hidden in the kinetics of reactions 23 and 25.

Hence, the (backward) translation from ODEs to structured models suitable for qualitative or stochastic analysis needs to be performed with great care. In [63], we look into the structure inference problem for ODEs models. We provide three biochemically relevant sufficient conditions under which the derived structure is unique and counterexamples showing the necessity of each of the following conditions.

- All reactions use pure mass/action kinetics, and the reaction parameters belong to a finite alphabet of symbols.
- The reaction network does not contain any void reaction.
- The same parameter is never used for two different reactions with the same reactants.

In summary, neither (4) nor (5) are suitable core notations for reaction networks, which may be subject to various complementary qualitative, stochastic, or continuous analysis techniques.

Structure versus list notation. Enumerating all reactions in a list (in fact an unordered set) is perfectly fine, if the notation is to be read by a computer. A structure-oriented graphical notation brings the additional advantage for an human reader of revealing the causality relation (structure, topology) among the reaction set. As such we consider (2) and (3) to be closer to the reality to be modelled.

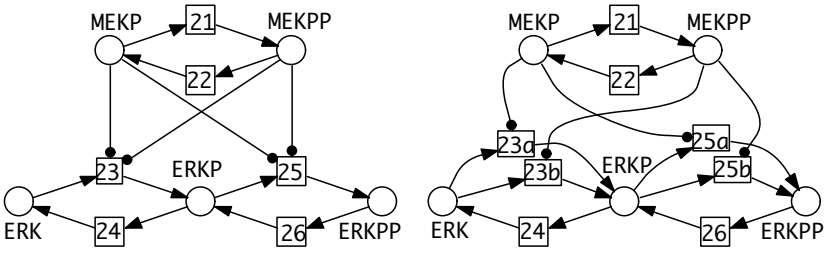


Fig. 3. Freestyle kinetics may hide essential structure. Assuming for reaction r_{23} in the network on the left the complex kinetics $v_{23} = v(r_{23}) = k_{23a} \cdot ERK \cdot MEKP + k_{23b} \cdot ERK \cdot MEKPP$, likewise for r_{25}, yields the network on right with all reactions having mass/action kinetics, e.g., $v_{23a} = v(r_{23a}) = k_{23a} \cdot ERK \cdot MEKP$.

Hypergraph versus Petri net. At first glance, both are equally suitable for a graphical representation of the network structure. The difference in the graphical notation style seems to be a matter of taste.

Petri nets enjoy a formal semantics, which could also be formulated in the terminology of hypergraphs. However, this might become a bit over-complicated. Reasoning about the reaction network structure, e.g. to derive behavioural properties, requires splitting a hyperarc into its ingredients. We need to refer to the reaction itself, its reactants – the nodes where the hyperarc starts, and its products – the nodes where the hyperarc goes to, and the individual multiplicities of those parts of the hyperarc in the case of stoichiometric information.

Moreover networks tend to be very large. This motivates the reuse of a well-established engineering principle to manage the design of large-scale systems – hierarchical decomposition. Formal concepts of hierarchical Petri nets with building blocks are well elaborated, see, e.g., [20]. They facilitate the modeling of large real-world systems because the comprehension of the whole network builds upon the understanding of all building blocks and the interconnection of their interfaces. Hierarchical structuring changes the style of representation, but does not change the actual net structure of the underlying reaction network.

Even if hierarchical modelling approaches have gained little popularity in practice so far in Systems Biology, they are indispensable for large-scale networks. There is nothing like this for hypergraphs.

This explains why we keep to Petri net terminology when in the next sections we briefly describe some contributions which Petri net theory has to offer for reaction network analysis, and thus to a Systems Biology toolkit.

Disclaimer: Being restricted to 20 pages, this paper can not exhaustively cover the field.

3 The Framework

In the following we recall our overall framework, introduced in [24], that relates the three major ways of modelling and analysing biochemical networks: qualitative, stochastic and continuous, compare Figure 4.

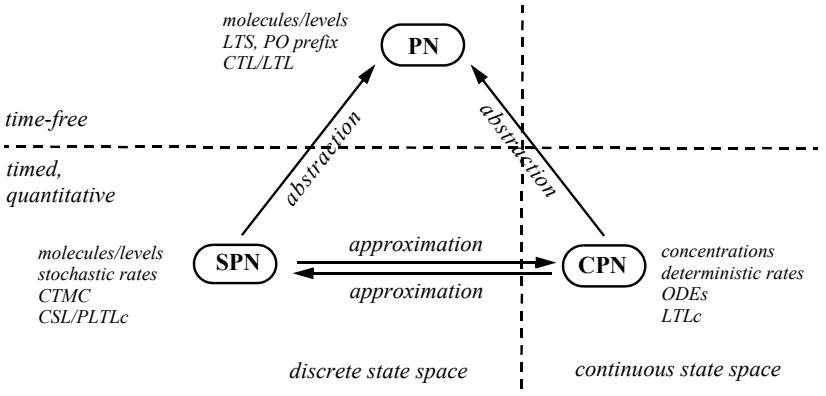


Fig. 4. Conceptual core framework

Qualitative paradigm. The most abstract representation of a biochemical network is *qualitative* and is minimally described by its topology, usually as a bipartite directed graph with nodes representing biochemical entities and reactions, or in Petri net terminology *places* and *transitions*. Arcs can be annotated with stoichiometric information, whereby the default stoichiometric value of 1 is usually omitted.

The qualitative description can be further enhanced by the abstract representation of discrete quantities of species, achieved in Petri nets by the use of tokens at places. These can represent the number of molecules, or the level of concentration, of a species, and a particular arrangement of tokens over a network is called a *marking*, specifying the system state.

The behaviour of such a net forms a discrete state space, which can either be captured as (1) a Labeled Transition System (LTS) (in the Petri net community better known as reachability or marking graph) to describe the net behaviour by all (totally ordered) interleaving sequences in the style of transition-labelled automata (interleaving semantics), or as (2) a finite prefix of a maximal branching process (PO prefix) to describe the net behaviour by all partially ordered transition sequences (partial order semantics). Animating a Petri net by sequentially firing individual transitions generates a path through the LTS.

Both descriptions of behaviour can be analysed for the purpose of model verification. In the bounded case, this is best done using model checking techniques, where the properties of interest are expressed by, e.g., a branching time temporal logic, one instance of which is Computational Tree Logic (CTL) [15], or a linear-time logic (LTL) [54].

The standard semantics for these qualitative Petri Nets (PN)[1] do not associate a time with transitions or the sojourn of tokens at places, and thus these descriptions are time-free. The qualitative analysis considers however all possible behaviour of the system under any timing. Thus, the qualitative Petri net model itself implicitly contains all possible time-dependent behaviour.

[1] As we want to use the term *Petri net* as umbrella term, we refer to the standard Petri net class as *qualitative Petri nets*, if required for the sake of unambiguity.

Timed information can be added to the qualitative description in two ways – stochastic and continuous.

Stochastic paradigm. The stochastic Petri net (SPN) description preserves the discrete state, but in addition associates an exponentially distributed firing rate (waiting time) with each reaction. The firing rates are typically state-dependent and specified by rate functions. All reactions, which occur in the PN, can still occur in the SPN, but their likelihood depends on the probability distribution of the associated firing rates. The underlying semantics is a Continuous-Time Markov Chain (CTMC). Stochastic simulation generates a random walk through the CTMC.

Special behavioural properties can be expressed using, e.g., Continuous Stochastic Logic (CSL), a stochastic counterpart of CTL which was originally introduced in [4], and extended in [5], or PLTLc, a probabilistic extension of LTL with constraints [18].

The PN is an abstraction of the SPN, sharing the same state space and transition relation with the stochastic model (if there are no parallel transitions), with the probabilistic information removed. All qualitative properties valid in the PN are also valid in the SPN, and vice versa.

Continuous paradigm. The Continuous Petri Net (CPN) replaces the discrete values of species with continuous values, and hence is not able to describe the behaviour of species at the level of individual molecules, but only the overall behaviour via concentrations. We can regard the discrete description of concentration levels as abstracting over the continuous description of concentrations. Timed information is introduced by the association of a particular deterministic firing rate with each transition, permitting the continuous model to be represented as a set of Ordinary Differential Equations (ODEs). The concentration of a particular species in such a model will have the same value at each point of time for repeated computational experiments. The state space of such models is continuous and linear. It can be analysed by, for example, Linear Temporal Logic with constraints (LTLc) in the manner of [11].

Moving between stochastic and continuous paradigms. One and the same quantitative model can be read either stochastically or continuously, no changes being required (up to some scaling in the rate functions for higher order reactions). In the stochastic case, the rate functions define the state-dependent rates of the individual occurrences of transition firings, while in the continuous case the rate functions define the strength of the state-dependent continuous flow. In [24] we discuss in more detail how the stochastic and continuous models are mutually related by approximation. The CPN replaces the probabilistically distributed reaction firing in the SPN by a particular average firing rate defining the continuous token flow. In turn, the stochastic model can be derived from the continuous model by approximation, reading the tokens as concentration levels, as introduced in [10].

Bridging to the continuous paradigm. It is well-known that time assumptions generally impose constraints on behaviour. The qualitative and stochastic models consider all possible behaviour under any timing, whereas the continuous

model is constrained by its inherent determinism to consider a subset. This may be too restrictive when modelling biochemical systems, which by their very nature exhibit variability in their behaviour.

The move from the discrete to the continuous paradigm may come along with counter-intuitive effects; e.g., a trap – a set of places which can never become empty in the discrete case as soon as it has obtained a token – can be emptied in the continuous case [62]. See [2] for another surprising example.

For illustration we briefly discuss the toy example given in Figure 2, left, which has been used in [61] to introduce the idea of Absolute Concentration Robustness (ACR). A place of a CPN (a variable of an ODEs model) is said to have ACR if its concentration is the same in all positive steady states; i.e., it does not depend on the total mass in the system, but only depends on the kinetic constants. The place A is such an ACR place; its steady state concentration is k_2/k_1, while B has the steady state concentration $total - k_2/k_1$, with $total$ being the conserved total mass in the network.

The place B forms a bad siphon, i.e., a siphon, not containing a trap. It is easy to see that the net does not have a live initial marking in the discrete world. Thus, there is always a dead state reachable, in which the bad siphon, i.e., B, is empty and all tokens reside in A, when the system is considered stochastically. However, there is no sign of the dead state when reading the net as CPN. Moreover, increasing the mass in the closed system will increase the steady value of B, as A has ACR. Thus, the continuous model may predict a simplified behaviour that does not reflect the variety, which is possible in the discrete case, e.g. in the given example, that the siphon will eventually get empty.

When analysing ODEs, one actually considers a family of related models, varying in some structural details, but generating the same ODEs. So far, the bridge to the continuous world is not equally well understood regarding the stochastic world. However, qualitative, stochastic & continuous Petri nets, which share structure should generally share some behavioural properties as well.

Applications. Case studies demonstrating the move between the three paradigms can be found in [29] (signalling cascade), [25] (biosensor gene regulation), and [28] (signal transduction network). In [23], we demonstrate by a case study the relationship between the strongly connected state space of the qualitative Petri net and the steady state behaviour of the corresponding continuous Petri net.

3.1 Beyond the Core Framework

In addition to the base case techniques, suitably expressive modelling formalisms and very efficient computational techniques are required to support abstraction levels and to give a holistic perspective on inherently multi-scale systems. As a consequence, a variety of different techniques have been proposed during the last decade, and new ones are constantly emerging. At the same time, many well-established and sophisticated modelling techniques from Computer Science, equipped with advanced analytical methods, have been applied to the exciting and challenging research field of Systems and Synthetic Biology.

Figure 5 shows the relationship between the most important modelling techniques contributed by the Petri net community. In the following we briefly characterize the main players, going beyond the core framework.

Extended Petri Nets (XPN) complement the standard ingredients of Petri nets by some dedicated features for the sake of easy modelling, but often sacrificing static analyzability. They exist in numerous variations. An extension particularly useful for Systems and Synthetic Biology are special arcs, e.g., read (test) arcs and inhibitor arcs, which provide adequate abstract modelling means for activating or inhibiting activities; see, e.g., [12, 35].

Functional Petri Nets (FPN) [38] pick up the idea of self-modifying nets [64] and use state-dependent functions to dynamically adjust arc multiplicities, which has been used to simulate metabolic pathways [14].

Time Petri Nets (TPN) equip transitions with a deterministic firing delay, typically characterized by a continuous time interval [49] of minimal and maximal delay. Remarkably, this net class can be fully analysed using a discretized state space only [56]; the particular capabilities for Systems and Synthetic Biology have be demonstrated in [58, 55, 57].

Generalised Stochastic Petri Nets (GSPN) extend SPN by immediate transitions [45], possibly complemented by special arcs. They have been used in, e.g., [42].

Extended Stochastic Petri Nets (XSPN) enrich GSPN with transitions having deterministic firing delays, typically characterized by a constant. They come in two flavours: relatively timed or absolutely timed, which helps in model-based wet-lab experiment design; see [32].

Hybrid Petri Nets (HPN) enrich CPN with discrete places and discrete transitions having deterministic firing delay, see, e.g., [17].

Hybrid Functional Petri Nets (HFPN) are a popular extension of HPN combining them with the self-modifying arcs of FPN. This net class is supported by the licensed tool Cell Illustrator, which has been deployed in many case studies [50].

Generalised Hybrid Petri Nets (GHPN) combine all features of XSPN and CPN [37].

In addition, further novel Petri net classes have been developed, which may turn out to be useful in Systems and Synthetic Biology. We briefly discuss two of them.

The *Probability Propagation Nets* [43] provide a Petri net representation for the propagation of probabilities and likelihoods in Bayesian networks. As such they enhance the transparency of propagation processes in the Bayesian world by exploiting structural and dynamic properties of Petri nets. They can be used to express knowledge which has been statistically extracted from gene expression array data (in the wet lab) to obtain executable and analyzable models of the data source (in the dry lab).

The *Error-correcting Petri Nets* [51] allow for the algebraic detection and modulo-p correction of non-reachable Petri net states, without ever constructing

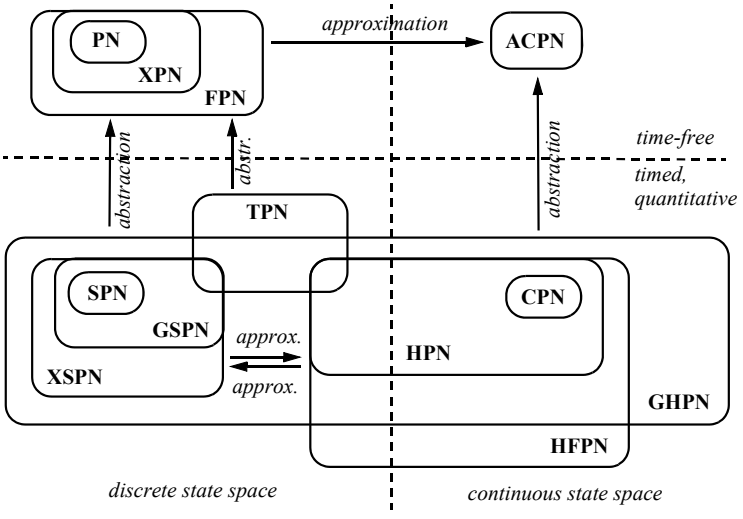


Fig. 5. Extended Framework showing the relation between some of the Petri net classes used in Systems and Synthetic Biology

the state space. They may gain importance in synthetic biology, e.g., to design biological systems with monitoring capabilities. The observation of states which are supposed to be non-reachable in the designed system will indicate some behavioural deviations from the original design. The method is able to pinpoint the source of the problem to a single place (several places) for a single error (multiple errors) depending on the amount of structural redundancy utilised during model construction.

This variety clearly demonstrates one of the big advantages of using Petri nets as a kind of umbrella formalism – the models may share the network structure, but vary in their quantitative (kinetic) information.

Related material. Formal definitions covering discrete, stochastic, continuous and hybrid variants of the Petri net formalisms, illustrated by many technical examples, can be found in [16]. The net classes closest to the continuous setting of biomolecular networks are discussed in Chapter 7, where the speeds (i.e., the rates) may depend on the C-marking (i.e., the current values of the continuous places). Combining the concept of variable rates with Definition 5.4 (feeding and draining speed of a place) and Definition 5.5 (the balance of the marking of a continuous place) explains how the ODEs model is set up.

However, the modelling complexity on hand calls for formal definitions and related tool support tailored to the specific needs of Systems and Synthetic Biology; e.g., by restricting the domain of the state-dependent rate functions to the pre-places of a transition in order to maintain a close relation between structure and behaviour. Biochemically interpreted Continuous Petri Nets have been introduced in [23], biochemically interpreted Stochastic Petri Nets in [24], and biochemically interpreted Extended Stochastic Petri Nets in [32]. More formal definitions covering the core framework can be found in [29].

4 Overview of Petri Net Specific Contributions

Probably the most popular notions in Petri net theory having their counterparts in Systems Biology are place and transition invariants (P/T-invariants for short). They are defined on a model's representation, which is called incidence matrix in the Petri net community and stoichiometric matrix in Systems Biology.

T-invariants can be seen as the integer counterparts of different representations of the continuous members of a cone, known in Systems Biology as elementary flux modes, extreme pathways, and generic pathways, while P-invariants correspond to conserved moieties [66].

Beyond that, Petri nets offer the following *additional features* to the list of established techniques known for quite a while in Systems Biology, such as nonlinear ODEs analysis, bifurcation theory, chemical reaction network theory [21], stoichiometric and flux balance analysis [52], etc..

4.1 Readability

Petri nets are an intuitive representation of reaction networks, which makes them easily comprehensible. They uniquely define derived notations, such as stoichiometric matrix or ODEs, but not vice versa; for details see Section 2.

Applications. Petri nets allow the unambiguous, but still readable representation of various types of biological processes at different levels of abstraction, with different resolution of detail, in one and the same model, ranging from the conformational change of a single molecule to the macroscopic response of a cell, the development of a tissue, or even the behaviour of a whole organism [48]. A recent survey [6] has shown how Petri nets can be applied to transcriptional, signalling, and metabolic networks, or combinations of them, illustrating this with a rich set of case studies. Most of these focus on the molecular level; however examples at the multi-cellular level include the signal-response behaviour of an organism [47], and developmental processes in multi-cellular pattern formation [7, 13].

4.2 Executability

Petri nets are directly executable with a suitable tool like Snoopy [59, 46]. The visualization of the token flow

- allows the user to better comprehend the net behaviour, and
- facilitates the communication between wet-lab experimentalists and dry-lab (computational) theoreticians.

Applications. The execution allows a time-free animation in the case of qualitative Petri nets (these are the standard Petri nets), and time-dependent animation and simulation in the case of quantitative Petri nets (stochastic, continuous, and hybrid Petri nets).

4.3 Causality

The structure of Petri nets reflects the causality relation among the reactions building the whole network, while clearly distinguishing between alternative and concurrent behaviour; see [31] for a more detailed discussion. This permits reasoning about "which reaction has to happen first, before another reaction can happen afterwards" in order to transport some mass (metabolic networks) or transfer some information (signal transduction network) through the network.

Applications. Reasoning over causality helps, e.g., in evaluating T-invariants (likewise elementary flux modes, extreme pathways, generic pathways), which define subnets in a reaction network. The unfolding of the involved reactions reveals their partial order causality, and may allow for complementary insights into the net behaviour; for examples see [29], [28].

4.4 Efficiency Gain

The Petri net structure allows for efficiency gains of some analysis algorithms, which are otherwise not possible. The crucial point is that the occurrence of a reaction only causes changes in its local environment, which consists of its pre-places, the reactants, and its post-places, the products. Vice versa, this requires that all dependencies are reflected in the structure. Thus we restrict the domain of transitions' rate functions to the transitions' pre-places.

Applications. Examples implemented in our toolkit, see Section 6, include:

- *Stochastic simulation* – to compute/update the state-dependent transition rates [34];
- *Static ordering of variables (places)* – to statically determine suitable orders of variables for high compression effects in symbolic representations, e.g., for symbolic model checking. In [60], we demonstrate how the performance of the popular CSL model checker PRISM [53] could take advantage of our structure-based heuristics to derive suitable orders of variables.
- *Static ordering of transitions* – to statically determine suitable orders of transitions for efficient symbolic state space construction, e.g., saturation-based construction, with the aim to avoid intermediate data structures which are much larger than the final ones.

4.5 Static Analysis

The standard analysis techniques, which are commonly used in many Computer Science approaches, need to construct the partial or full state space (expressed as LTS or PO prefix). In addition to these dynamic analysis techniques, Petri net theory offers a variety of static analysis techniques, which permit the decisions of general behavioural properties, such as boundedness and liveness, without state space construction.

- *Boundedness* ensures upper bounds for all species, which implies a finite state space. Structural boundedness guarantees this behaviour for any initial marking.
- *Liveness* ensures that all reactions will forever contribute to the system behaviour, which precludes dead states. Structural liveness guarantees that there is a live initial marking.

Static analysis techniques, which we found useful so far in analyzing biochemical reaction networks, comprise:

- *property-preserving reduction rules*, which replace subnetworks by generally smaller ones without changing the general behavioural properties;
- *structural analyses*, which determine, e.g., the net structure classes (such as state machine, synchronisation graph, free choice nets, extended simple nets), Siphon Trap Property (STP), or rank theorem, and apply related theorems;
- *linear programming techniques*, which perform, e.g., structural boundedness check to decide if the state space is finite; P/T-invariant computation and coverage test, which supports also model validation; or state/trap equation check as sufficient criteria for non-reachability.

More details can be found in [29], where we recall basic notions and theorems of Petri net theory. Most of them are implemented in our Petri net analysis tool Charlie [22].

Applications. Case studies demonstrating the power of static analysis techniques can be found in [29] (signalling cascade), [25] (biosensor gene regulation), and [28] (signal transduction network).

The comprehensive textbook [52] is focused on the stoichiometric matrix and related analysis techniques. It is also a good entry point for the growing body of related literature. The use of P/T-invariants for the validation of biochemical networks has been introduced to the Petri net community in [30].

Specific application examples include:

- Angeli et al. introduce in [3] a sufficient criterion for persistency of ODEs models, i.e., of continuous Petri nets, which is based on the structure. It ensures persistency for reaction networks with mass/action kinetics and arbitrary parameters. This structural criterion closely resembles a famous liveness criterion of Petri net theory – the Siphon Trap Property, adjusted to the continuous case: the trap is replaced by a P-invariant.
- In [27], we use T-invariants and the related notion of Abstract Dependent Transition sets (ADT sets) to automatically derive hierarchically structured networks.
- In [36], we apply structural analysis, specifically T-invariants, to reduce a given ODEs model, while preserving the essential steady state behaviour. The structural analysis identifies the core network and its fragile node (robustness analysis).

– In [33], we shed new light on the importance of the Siphon Trap Property for continuization (fluidization), which is determined for a number of biochemical networks from our repository.

We expect further results along this line in the future. Petri nets offer a suitable framework for reasoning about the relationship between the three basic modelling paradigms. For this purpose, our tool Snoopy [59] allows the user to move easily between the three worlds, and thus ensuring the equivalence of the related models.

5 Model Checking

Model checking is a powerful analysis approach, well-established in Computer Science and now gaining increasing popularity in Systems and Synthetic Biology, which can be performed in the three modelling paradigms. It permits checking the behaviour of a model for certain properties expressed in temporal logics. These are unambiguous languages, which provide flexible formalisms to check the validity of statements (propositions) in relation to the execution of the model. Temporal logics come in different flavours; for example branching-time and linear-time logics. The latter are more attractive to biologists who are used to reason about time series behaviour.

Analytical model checking is usually restricted to LTS descriptions (interleaving semantics). However, we can take advantage of Petri nets' partial order semantics to perform model checking over behaviour descriptions given as PO prefix, which preserve the difference between alternative and concurrent occurrences of reactions. The power of this technique for Systems and Synthetic Biology has not been systematically explored yet.

Analytical model checking may become computationally expensive and generally requires network boundedness. An efficient alternative approach is *simulative model checking*, where analysis is performed over time series traces generated by model simulation, which works both for bounded and unbounded models. It involves two approximations: a finite number of finite simulation traces is only considered. Simulative model checking can perhaps best be characterised as behaviour checking, being applicable not only to continuous or stochastic models, but also to any time series data. This approach can thus be used to compare models with physical systems in terms of data generated by simulations of a model and laboratory experiments.

In [29,28], we have demonstrated how to perform model checking in the three modelling paradigms in a concerted manner.

5.1 Characterising Biochemical Species' Behaviour

The behaviour of biochemical species can be described for quantitative model checking using four distinct descriptive approaches, with increasing specificity; qualitative, semi-qualitative, semi-quantitative and quantitative [18].

Qualitative formulae use derivatives of biochemical species concentrations or mass, determined by the function $d(.)$, and the temporal operators U, F, G to describe the general trend of the behaviour. Semi-qualitative extend qualitative with relative concentrations using the function $max(.)$, which yields the maximal value of a species observed in the whole run. Semi-quantitative extend semi-qualitative with absolute time values by referring to the predefined system variable *time*. Finally, quantitative extend semi-quantitative with absolute concentration values.

For example, the transient activation of a biochemical species called *Protein* can be expressed in these approaches with increasing accuracy. The following queries determine the probabilities of the statements to be true using PLTLc.

Qualitative. Protein rises then falls:

$$\mathbf{P}_{=?}\ [\ d(\text{Protein})\ >\ 0\ U\ (\ G(\ d(\text{Protein})\ <\ 0\ )\ )\ ].$$

Semi-qualitative. Protein rises then falls to less than 50% of its peak concentration:

$$\mathbf{P}_{=?}\ [\ (\ d(\text{Protein})\ >\ 0\ )\ U\ (\ G(\ d(\text{Protein})\ <\ 0\ )\ \wedge\ F(\ [\text{Protein}]\ <\ 0.5 * max[\text{Protein}]\ )\ )\ ].$$

Semi-quantitative. Protein rises then falls to less than 50% of its peak concentration at 60 minutes:

$$\mathbf{P}_{=?}\ [\ (\ d(\text{Protein})\ >\ 0\ )\ U\ (\ G(\ d(\text{Protein})\ <\ 0\ )\ \wedge\ F(\ \text{time}\ =\ 60\ \wedge\ \text{Protein}\ <\ 0.5 * max(\text{Protein})\ )\ )\ ].$$

Quantitative. Protein rises then falls to less than $100\mu Mol$ at 60 minutes:

$$\mathbf{P}_{=?}\ [\ (\ d(\text{Protein})\ >\ 0\ )\ U\ (\ G(\ d(\text{Protein})\ <\ 0\ )\ \wedge\ F(\ \text{time}\ =\ 60\ \wedge\ \text{Protein}\ <\ 100\ )\ )\ ].$$

5.2 Applications for Model Checking

There are five major areas of applications for model checking in Systems and Synthetic Biology [8].

Model validation. Does the model behave in the way we expect?

Model comparison. How similar are two models, independent of their underlying formalisms?

Model searching. In a database of alternative models, which of them exhibit a particular property? This can be used to select among competing descriptions of a system produced by various research groups. Given a large database of models (either a general collection or variants of the same model), one can use model behaviour checking to perform systematic database queries, such as "find all models that show oscillatory behaviour under some conditions"

or "find all descriptions of this signaling pathway that are transiently active after growth factor stimulation".

Model analysis. In a collection of structure-preserving variants of a model (e.g., different concentrations of species representing in-silico gene knock-outs), which models show a certain behaviour? E.g., how many knock-outs lead to a loss of oscillating behaviour?

Model construction. Which modifications of a model lead to a desired property? Modifications can involve changes in kinetic parameters or initial concentrations, but they can also be more complex, for example changing the topology of the model by removing or even adding new components. How to do this efficiently is still an active area of research.

6 Tools

BioModel Engineering of non-trivial case studies requires adequate tool support. We provide a sophisticated toolkit covering the whole framework; publicly available at `http://www-dssz.informatik.tu-cottbus.de`.

Snoopy – tool for modelling and animation/simulation of hierarchically structured graphs, among them qualitative, stochastic, and continuous Petri nets [59], [46], recently extended by coloured counterparts of these net classes [44] and Generalised Hybrid Petri Nets, supporting dynamic partitioning [37].

Charlie – multi-threaded analysis tool of standard Petri net properties and techniques of Petri net theory including structural boundedness check, P/T-invariants, STP, rank theorem, structural reduction. It also supports explicit CTL and LTL model checking [22], mainly for teaching purposes.

Marcie – is built upon an Interval Decision Diagram (IDD) engine and supports, besides the analysis of standard Petri net properties (boundedness, dead states, liveness, reversibility) symbolic CTL model checking of qualitative Petri nets [35] and multi-threaded symbolic CSL model checking of Generalised Stochastic Petri Nets [60], recently extended by rewards. Both model checkers outperform comparable tools on the market for benchmarks of typical biochemical networks [34].

Exact analyses of bounded models are complemented by approximative model checking, which is also suitable for unbounded models. Two approximative engines support fast adaptive uniformisation and distributed Gillespie simulation.

In addition, Snoopy files are directly read by ADAM – a tool applying computational algebra to analyse the dynamics of discrete models [65], and simulation traces generated with Snoopy are read by MC2(PLTLc) – a Monte-Carlo Model Checker for PLTLc [18]. Additionally, Snoopy provides export to several foreign analysis tools, among them Anastasia [67] for efficient computation of bad

siphons and STP decision, and SBML import/export, which opens the door to a bunch of tools popular in Systems and Synthetic Biology.

7 Outlook and Challenges

A drawback of current modelling approaches, including Petri nets, are their limitation to relatively small networks. Thus overall, one of the major challenges lies in the application of Petri nets to achieve richer descriptions of biological systems than are currently practicable. This includes modelling compartments, locality and mobility of players at the *intracellular level*. Biological systems can be represented as networks which themselves typically contain regular (network) structures, and/or repeated occurrences of network patterns. This organisation occurs in a hierarchical manner, reflecting the physical and spatial organisation of the organism, from the intracellular to the intercellular level and beyond (tissues, organs etc.). Examples of organisation at the intracellular level include: cytosol, nucleus, ribosome, endoplasmic reticulum, Golgi apparatus, and mitochondria.

Further challenges lie in modelling at the *intercellular level*, for examples societies of single-celled organisms (including mixtures of organisms), or multicellular organisms at the tissue or organ level. There are many examples of communication in multicellular organisms including endocrine, juxtacrine, etc.

Although such network models can be designed using standard Petri nets, so far there is no dedicated support for such structuring; it becomes impractical as the size of the networks to be modelled increases. Besides the purely technical aspect due to the impracticality of handling large flat nets, humans need some hierarchy and organisation in what they are designing in order to be able to conceptualise the modelled object. Thus, models should explicitly reflect the organisation in complex biological systems.

There are two established orthogonal concepts in Petri net modelling to manage large-scale networks - structuring by hierarchies and colours. However, their potential for BioModel Engineering has not been systematically explored so far. In order to cope with the challenge of modelling at such a variety of levels, we need expressive modelling techniques embedded as part of a robust BioModelling Engineering approach, supporting, e.g., model version control in a collaborative environment for model construction.

8 Summary

Petri nets are a natural and established notation for describing reaction networks because both share the bipartite property. We have shown that Petri nets can be used to perform all major modelling and simulation approaches central to Systems Biology. Thus, they may serve as a kind of umbrella formalism integrating qualitative and quantitative (i.e. stochastic, continuous, or hybrid) modelling and analysis techniques.

However, as we have indicated, there are several important open challenges posed by modelling biological systems which need to be addressed.

References

1. Aderem, A.: Systems Biology: Its Practice and Challenges. Cell 121(4), 511–513 (2005)
2. Angeli, D.: Boundedness analysis for open Chemical Reaction Networks with mass-action kinetics. J. Natural Computing (2009), doi:10.1007/s11047-009-9163-7
3. Angeli, D., De Leenheer, P., Sontag, E.D.: A Petri net approach to the study of persistence in chemical reaction networks. Mathematical Biosciences 210(2), 598–618 (2007)
4. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model checking continuous time Markov chains. ACM Trans. on Computational Logic 1(1) (2000)
5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model checking algorithms for continuous-time Markov chains. IEEE Trans. on Software Engineering 29(6) (2003)
6. Baldan, P., Cocco, N., Marin, A., Simeoni, M.: Petri Nets for Modelling Metabolic Pathways: a Survey. J. Natural Computing (9), 955–989 (2010)
7. Bonzanni, N., Krepska, E., Feenstra, K., Fokkink, W., Kielmann, T., Bal, H., Heringa, J.: Executing multicellular differentiation: quantitative predictive modelling of c. elegans vulval development. Bioinformatics 25, 2049–2056 (2009)
8. Breitling, R., Donaldson, R.A., Gilbert, D., Heiner, M.: Biomodel Engineering - From Structure to Behavior. Lect. Notes Bioinformatics 5945, 1–12 (2010)
9. Brightman, F.A., Fell, D.A.: Differential feedback regulation of the MAPK cascade underlies the quantitative differences in EGF and NGF signalling in PC12 cells. FEBS letters 482(3), 169–174 (2000)
10. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using continuous time Markov chains. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 44–67. Springer, Heidelberg (2006)
11. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: Machine learning biochemical networks from temporal logic properties. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 68–94. Springer, Heidelberg (2006)
12. Chaouiya, C.: Petri Net Modelling of Biological Networks. Briefings in Bioinformatics 8(4), 210 (2007)
13. Chen, L., Masao, N., Kazuko, U., Satoru, M.: Simulation-based model checking approach to cell fate specification during c. elegans vulval development by hybrid functional Petri net with extension. BMC Systems Biology 42 (2009)
14. Chen, M., Hariharaputran, S., Hofestädt, R., Kormeier, B., Spangardt, S.: Petri net models for the semi-automatic construction of large scale biological networks. J. Natural Computing (2009), doi:10.1007/s11047-009-9151-y
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (2001)
16. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer, Heidelberg (2010)
17. Doi, A., Drath, R., Nagaska, M., Matsuno, H., Miyano, S.: Protein Dynamics Observations of Lambda-Phage by Hybrid Petri net. Genome Informatics, 217–218 (1999)
18. Donaldson, R., Gilbert, D.: A model checking approach to the parameter estimation of biochemical pathways. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 269–287. Springer, Heidelberg (2008)

19. Endy, D.: Foundations for engineering biology. Nature 438(7067), 449–453 (2005)
20. Fehling, R.: A concept of hierarchical Petri nets with building blocks. In: Rozenberg, G. (ed.) APN 1993. LNCS, vol. 674, pp. 148–168. Springer, Heidelberg (1993)
21. Feinberg, M.: Mathematical aspects of mass action kinetics. In: Lapidus, L., Amundson, N.R. (eds.) Chemical Reactor Theory: A Review, ch.1, pp. 1–78. Prentice-Hall, Englewood Cliffs (1977)
22. Franzke, A.: Charlie 2.0 - a multi-threaded Petri net analyzer. Diploma Thesis, Brandenburg University of Technology at Cottbus, CS Dep. (2009)
23. Gilbert, D., Heiner, M.: From Petri nets to differential equations - an integrative approach for biochemical network analysis. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 181–200. Springer, Heidelberg (2006)
24. Gilbert, D., Heiner, M., Lehrack, S.: A unifying framework for modelling and analysing biochemical pathways using petri nets. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 200–216. Springer, Heidelberg (2007)
25. Gilbert, D., Heiner, M., Rosser, S., R., Fulton, X.G., Trybiło, M.: A Case Study in Model-driven Synthetic Biology. In: Proc. 2nd IFIP Conference on Biologically Inspired Collaborative Computing (BICC), IFIP WCC 2008, Milano, pp. 163–175 (2008)
26. Heinemann, M., Panke, S.: Synthetic biology - putting engineering into biology. Bioinformatics 22(22), 2790–2799 (2006), http://dx.doi.org/10.1093/bioinformatics/btl469
27. Heiner, M.: Understanding Network Behaviour by Structured Representations of Transition Invariants – A Petri Net Perspective on Systems and Synthetic Biology. Natural Computing Series, pp. 367–389. Springer, Heidelberg (2009), http://www.springerlink.com/content/m8t30720r141442m
28. Heiner, M., Donaldson, R., Gilbert, D.: Petri Nets for Systems Biology. In: Iyengar, M.S. (ed.) Symbolic Systems Biology: Theory and Methods, ch.21, Jones and Bartlett Publishers, Inc. (2010)
29. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets in systems and synthetic biology. In: Bernardo, M., Degano, P., Tennenholtz, M. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
30. Heiner, M., Koch, I.: Petri Net Based Model Validation in Systems Biology. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 216–237. Springer, Heidelberg (2004)
31. Heiner, M., Koch, I., Will, J.: Model Validation of Biological Pathways Using Petri Nets - Demonstrated for Apoptosis. BioSystems 75, 15–28 (2004)
32. Heiner, M., Lehrack, S., Gilbert, D., Marwan, W.: Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. Transactions on Computational Systems Biology XI, 138–163 (2009)
33. Heiner, M., Mahulea, C., Silva, M.: On the Importance of the Deadlock Trap Property for Monotonic Liveness. In: Int. Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri Nets 2010 (2010)
34. M., Heiner, C.R., M., Schwarick, S.S.: A Comparative Study of Stochastic Analysis Techniques. In: Proc. CMSB, pp. 96–106 (2010) ; ACM Digital Library 978-1-4503-0068-1/10/09
35. Heiner, M., Schwarick, M., Tovchigrechko, A.: DSSZ-MC - A Tool for Symbolic Analysis of Extended Petri Nets. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 323–332. Springer, Heidelberg (2009)
36. Heiner, M., Sriram, K.: Structural analysis to determine the core of hypoxia response network. PLoS ONE 5(1), e8600 (2010),

http://dx.doi.org/10.1371/journal.pone.0008600, doi:10.1371/journal.pone.0008600
37. Herajy, M., Heiner, M.: Hybrid Petri Nets for Modelling of Hybrid Biochemical Interactions. In: Proc. AWPN 2010, CEUR Workshop Proceedings, vol. 643, pp. 66–79. CEUR-WS.org (2010)
38. Hofestädt, R., Thelen, S.: Quantitative modeling of biochemical networks. In Silico Biol. 1, 39–53 (1998)
39. Hood, L.: Systems biology: integrating technology, biology, and computation. Mechanisms of Ageing and Development 124(1), 9–16 (2003)
40. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., Kummer, U.: Copasi – a complex pathway simulator. BioInformatics 22(24), 3067–3074 (2006)
41. Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., et al.: The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models. J. Bioinformatics 19, 524–531 (2003)
42. Lamprecht, R., Smith, G.D., Kemper, P.: Stochastic Petri net models of Ca^{2+} signaling complexes and their analysis. J. Natural Computing (2009), doi:10.1007/s11047-009-9143-y
43. Lautenbach, K., Pinl, A.: A Petri net representation of Bayesian message flows: importance of Bayesian networks for biological applications. J. Natural Computing (2009), doi:10.1007/s11047-009-9142-z
44. Liu, F., Heiner, M.: Computation of Enabled Transition Instances for Colored Petri Nets. In: Proc. AWPN 2010, CEUR Workshop Proceedings, vol. 643, pp. 51–65. CEUR-WS.org (2010)
45. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets, 2nd edn. John Wiley and Sons, Chichester (1995)
46. Marwan, W., Rohr, C., Heiner, M.: Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks. Methods in Molecular Biology, vol. ch.21. Humana Press (2011)
47. Marwan, W., Sujathab, A., Starostzik, C.: Reconstructing the regulatory network controlling commitment and sporulation in *Physarum polycephalum* based on hierarchical Petri net modeling and simulation. J. of Theoretical Biology 236(4), 349–365 (2005)
48. Marwan, W., Wagler, A., Weismantel, R.: Petri nets as a framework for the reconstruction and analysis of signal transduction pathways and regulatory networks. J. Natural Computing (2009), doi:10.1007/s11047-009-9152-x
49. Merlin, P.: A Study of the Recoverability of Computing Systems. Irvine: Univ. California, PhD Thesis, available from Ann Arbor: Univ Microfilms, No. 75–11026 (1974)
50. Nagasaki, M., Saito, A., Doi, A., Matsuno, H., Miyano, S.: Foundations of Systems Biology Using Cell Illustrator and Pathway Databases. Computational Biology, vol. 13. Springer, Heidelberg (2009)
51. Pagnoni, A.: Error-correcting Petri nets. J. Natural Computing (2009), doi:10.1007/s11047-009-9150-z
52. Palsson, B.O.: Systems Biology: Properties of Reconstructed Networks. Cambridge University Press, Cambridge (2006)
53. Parker, D., Norman, G., Kwiatkowska, M.: PRISM 3.0.beta1 Users' Guide (2006)
54. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77), pp. 46–57. IEEE Computer Society Press, Providence (1977)

55. Popova, L., Heiner, M.: Quantitative evaluation of time petri nets and applications to technical and biochemical networks. In: Proc. Int. Workshop on Concurrency, Specification and Programming (CS&P 2007), Lagów, vol. 2, pp. 473–484 (2007)
56. Popova-Zeugmann, L.: On time Petri nets. J. Inform. Process. Cybern. EIK 27(4), 227–244 (1991)
57. Popova-Zeugmann, L.: Quantitative evaluation of time-dependent Petri nets and applications to biochemical networks. J. Natural Computing (2010), doi:10.1007/s11047-010-9211-3
58. Popova-Zeugmann, L., Heiner, M., Koch, I.: Time Petri Nets for Modelling and Analysis of Biochemical Networks. Fundamenta Informaticae 67 (2005)
59. Rohr, C., W., Marwan, M.H.: Snoopy - a unifying Petri net framework to investigate biomolecular networks. Bioinformatics 26(7), 974–975 (2010)
60. Schwarick, M., Heiner, M.: CSL Model Checking of Biochemical Networks with Interval Decision Diagrams. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 296–312. Springer, Heidelberg (2009)
61. Shinar, G., Feinberg, M.: Structural Sources of Robustness in Biochemical Reaction Networks. Science 327, 1389–1391 (2010)
62. Silva, M., Recalde, L.: Petri nets and integrality relaxations: A view of continuous Petri net models. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 32(4), 314–327 (2002)
63. Soliman, S., Heiner, M.: A unique transformation from ordinary differential equations to reaction networks. PlosONE 5(12), e14284 (2010)
64. Valk, R.: Self-modifying nets, a natural extension of Petri nets. In: Automata, Languages and Programming. LNCS, vol. 62, pp. 464–476. Springer, Heidelberg (1978)
65. Veliz-Cuba, A., Jarrah, A.S., Laubenbacher, R.: Polynomial algebra of discrete models in systems biology. Bioinformatics 26(13), 1637–1643 (2010)
66. Wagler, A., Weismantel, R.: The combinatorics of modeling and analyzing biological systems. J. Natural Computing (2009), doi:10.1007/s11047-009-9165-5
67. Wimmel, H.: Anastasia – computing traps, siphons, and more (2010), `http://service-technology.org/anastasia`

State Estimation and Fault Detection Using Petri Nets

Alessandro Giua

DIEE, University of Cagliari, Piazza d'Armi, 09123 Cagliari, Italy
giua@diee.unica.it
http://www.diee.unica.it/giua

Abstract. This extended abstract serves as a support for the plenary address given by the author at the 32nd International Conference on Application and Theory of Petri Nets and Concurrency.

1 State Estimation and Observability

The *state estimation problem* for a dynamical system consists in reconstructing the current and past values of its internal states from the knowledge of the current and past values of its external measurable outputs. If such a problem admits a solution, the system is said to be *observable*.

Observability is a fundamental property that has received a lot of attention in the framework of time driven systems (TDS), given the importance of reconstructing plant states that cannot be measured for monitoring and for implementing state feedback control laws. Although less popular in the case of discrete event systems (DES), the problem of state estimation has been discussed in the literature and has generated several original issues.

A first issue in this framework consists in the definition of a suitable output. In fact, TDS are naturally described using the paradigm of input–state–output, the outputs being algebraic functions of the states. On the contrary, DES are typically described in terms of states and events and it may possible to associate labels to states or to events (e.g., as in a Moore or Mealy machine). When a system is in a given state, or when an event occurs, the corresponding label is observed. Hence this suggests that there are at least two possible ways of defining the outputs of a DES. The most popular choice is that of considering event occurrences as outputs. This talk will mostly focus on approaches of this type.

As a second issue, let us observe that there have been two main approaches to state estimation for DES.

- The first approach, that we call *control theory approach*, assumes that systems are deterministic, i.e., the labeling function that assigns to each event a label is injective. This means that any event occurrence can be precisely detected: the only uncertainty in this case is the lack of information on the initial state of the system. This setting is close to the state estimation problem in TDS.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 38–48, 2011.

– The second approach, that we call *computer science approach*, assumes that systems are non deterministic, i.e., the labeling function that assigns to each event a label is non-injective and possibly erasive. This means that an event occurrence may not be distinguishable from the occurrence of another different event or may even remain undetected. This adds an additional degree of complexity to the problem of state estimation. Furthermore, it also poses a new problem of *sequence estimation* (as opposed to state estimation), i.e., the problem of reconstructing the sequence of events that has occurred.

As a third issue, we mention that while the state estimation problem in TDS consists in producing an estimate $\hat{x}(t)$ of the actual state $x(t)$ at time t, in the DES framework what is actually computed is an enumerable set of *consistent states*, i.e., states in which the system may be given the observed output.

A fourth issue consists in the fact that in a DES setting the state estimation problem is often solved with a bounded delay, i.e., due to nondeterminism an observer may not be able to correctly estimate the current system state but may be able to reconstruct it only after a given number of observations have elapsed.

Traditionally the problem of state estimation for DES has been addressed using automata models, or related models of computation. For a DES modeled as nondeterministic finite automata, the most common way of solving the problem of partial observation is that of converting, using a standard *determinization* procedure, the nondeterministic finite automaton (NFA) into an equivalent deterministic finite automaton (DFA) where: (i) each state of the DFA corresponds to a set of states of the NFA; (ii) the state reached on the DFA after the word w is observed, gives the set of *states consistent with the observed word* w.

We first observe that an analogous determinization procedure as that used in the case of automata, cannot be used in the Petri net (PN) framework. In fact, a nondeterministic PN cannot be converted into an equivalent deterministic PN, because of the strict inclusions:

$$\mathcal{L}_{det} \subsetneq \mathcal{L} \subsetneq \mathcal{L}_{\lambda}$$

where

– $\mathcal{L}_{det}$ is the set of deterministic PN languages;
– $\mathcal{L}$ is the set of λ-free PN languages, namely, languages accepted by nets where no transition is labeled with the empty string;
– $\mathcal{L}_{\lambda}$ is the set of arbitrary PN languages where a transition may also be labeled with the empty string.

Furthermore, the main drawback of the automata based approaches is the requirement that the set of consistent states must explicitly be enumerated.

In this talk several approaches for the state estimation of PN models will be discussed and it will be shown that they offer several computational advantages with respect to the automata approaches. In particular the fact that the state of a net, i.e., its marking, is a vector and that the state space does not require to be enumerated is a major asset and paves the way for the development of efficient linear algebraic tools for state estimation.

2 Fault Detection and Diagnosability

A *fault* is defined to be any deviation of a system from its normal or intended behavior. *Diagnosis* is the process of detecting an abnormality in the system's behavior and isolating the cause or the source of this abnormality.

Failures are inevitable in today's complex industrial environment and they could arise from several sources such as design errors, equipment malfunctions, operator mistakes, and so on. The need of automated mechanisms for the timely and accurate diagnosis of failures is well understood and appreciated both in industry and in academia.

In this framework two different problems can be solved: the problem of *diagnosis* and the problem of *diagnosability*. Solving a problem of diagnosis means that we associate to each observed evolution of the outputs a diagnosis state, such as "normal" or "faulty" or "uncertain". Solving a problem of diagnosability is equivalent to determine if the system is diagnosable, i.e., to determine if, once a fault has occurred, the system can detect its occurrence with an observation of bounded length.

Failure detection problems have been addressed by several communities. Within systems theory, both in the framework of TDS and DES the approaches for solving these problems have often been based on methodologies developed to cope with state estimation and sequence estimation. Recent approaches to diagnosis and diagnosability of Petri net models will be discussed in the second part of this talk.

3 Relevant Literature for State Estimation and Fault Detection of Discrete Event Systems

3.1 State Estimation Using Automata

For systems represented as finite automata, Ramadge [32] was the first to show how an observer could be designed for a partially observed system.

Caines *et al.* [9] showed how it is possible to use the information contained in the past sequence of observations (given as a sequence of observation states and control inputs) to compute the set of states consistent with observation. In [10] the observer output is used to steer the state of the plant to a desired terminal state. The approach of Caines is based on the construction of an observer tree to determine the set of markings consistent with the observed behavior. A similar approach was also used by Kumar *et al.* [26] when defining observer based dynamic controllers in the framework of supervisory predicate control problems.

Özveren and Willsky [30] proposed an approach for building observers that allows one to reconstruct the state of finite automata after a word of bounded length has been observed, showing that an observer may have an exponential number of states.

Several works dealt with state and sequence estimation in a decentralized setting with communications.

In [43] Tripakis defined a property that he calls *local observability*. The idea is the following: a set of local sites observe, through their own projection masks, a word w of symbols that is known to belong to a language L. A language $K \subset L$ is locally observable if, assuming all local sites send to a coordinator all observed strings, the coordinator can decide for any w if the word belongs to K or to $L \setminus K$. This property was shown to be undecidable even when languages L and K are regular: this is due to the fact that the length of a word w can be arbitrarily long.

A very general approach for observability with communication has been presented by Barret and Lafortune [1] in the context of supervisory control, and several techniques for designing a possibly optimal communication policy have also been discussed therein. By optimal we mean that the local sites communicate as late as possible, only when strictly necessary to prevent the undesirable behavior.

In the previous approach communications are decided by the local observers and are triggered by the observation of an event. Ricker and Caillaud [34] have considered a setting where communications may also be triggered by the receiver, that requests information from a sender. Furthermore, they also discuss policies where communication occurs after prefixes of any of the behaviors involved in a violation of co-observability, not just those that may result in undesired behavior.

A dual problem, strictly related to observability, is *opacity*. A system is (current-state) opaque if an observer cannot determine if the (current) state of the system certainly belongs to a given set of secret states. See the work of Saboori and Hadjicostis [37,38,39] and of Dubreil *et al.* [15].

Finally, we also mention that several other works have addressed the related issue of control under partial observations. However, they are not mentioned here because only partially relevant to the problem of state estimation.

3.2 State Estimation Using Petri Nets

Our group was one of the first to address the problem of state estimation for Petri nets and has explored different methodologies.

In [18] some preliminary concepts dealing with the observability properties of nets where all transitions are observable have been introduced. As a extension of this work, in [20] a procedure that produces an estimate of the state was proposed, while the special structure of Petri nets allows one to determine, using linear algebraic tools, if a given marking is consistent with the observed behavior without the explicit enumeration of the (possibly infinite) consistent set. This approach was also successfully applied to timed nets: a procedure to exploit the timing information to improve the marking estimate was given in [21].

In [19] λ-free labeled PNs, i.e., PNs where no transition is labeled with the empty string, were studied. The restrictive assumption was that nondeterministic transitions are *contact-free*, i.e., for any two nondeterministic transitions t and t' the set of input and output places of t cannot intersect the set of input and output places of t'. In this case it is possible to give a linear algebraic characterization of the set of consistent markings that depends on some parameters that can be recursively computed.

Nets with unobservable transitions, i.e., transitions labeled with the empty string, were studied in [12]. Here we introduced the notion of *basis marking*. The idea is that under very general conditions, namely the acyclicity of the unobservable subnet, it is possible to characterize the set of markings consistent with an observation in terms of sequences of minimal length. The markings reached by these sequences are called basis markings and all other markings consistent with the observation can be obtained from the knowledge of this smaller set.

A similar approach that also deals with distributed implementation was presented by Jiroveanu *et al.* in [25]. The problem of finding upper bounds on the number of markings that are consistent with an observed sequence of labels was addressed by Ru and Hadjicostis in [35]. Here they show that the number of consistent markings, i.e., the complexity of solving a state estimation problem, in a Petri net with nondeterministic transitions is at most polynomial in the length of the observation sequence, though it remains exponential in certain parameters of the net.

Finally a different approach was proposed by Meda–Campaña *et al.* [29], using interpreted Petri nets to model the system and the observer. The main idea is to start the observer with a marking bigger than the real one and then eliminate some tokens until the observer and system markings are equal. Interpreted Petri nets using a mix of transition firings and place observations have also been used by Ramírez-Treviño *et al.* in [33] where it was shown that observability defined as in [29] is equivalent to observability in [18] and it was shown how to construct an observer for binary interpreted Petri nets when the observability property is verified.

3.3 Diagnosis and Diagnosability Using Automata

In the contest of DES several original theoretical approaches have been proposed using automata.

Lin [28] proposed a state-based DES approach to failure diagnosis. The problems of off-line and on-line diagnosis are addressed separately and notions of diagnosability in both of these cases are presented. The author gives an algorithm for computing a diagnostic control, i.e., a sequence of test commands for diagnosing system failures. This algorithm is guaranteed to converge if the system satisfies the conditions for on-line diagnosability.

The group of Lafortune [40,41,13] in a series of papers proposed a seminal approach to failure diagnosis. The system is modeled as an automaton in which failures are modeled by a (possibly strict) subset of the set of unobservable events. The level of detail in a discrete event model appears to be quite adequate for a large class of systems and for a wide variety of failures to be diagnosed. The approach is applicable whenever failures cause a distinct change in the system status but do not necessarily bring the system to a halt. In [40] and [41] Sampath *et al.* provided a systematic approach to solve the problem of diagnosis using *diagnosers*. They also gave a formal definition of diagnosability in the framework of formal languages and established necessary and sufficient conditions for diagnosability of systems.

Hashtrudi Zad *et al.* [23] presented a state-based approach for on-line passive fault diagnosis. In this framework, the system and the fault detection system do not have to be initialized at the same time. Furthermore, no information about the state or even the condition (failure status) of the system before the initiation of diagnosis is required. The design of the fault detection system, in the worst case, has exponential complexity. A model reduction scheme with polynomial time complexity is introduced to reduce the computational complexity of the design. Diagnosability of failures is studied, and necessary and sufficient conditions for failure diagnosability are derived.

We conclude mentioning several works dealing with diagnosis in a decentralized settings, possibly with communications or with a coordinator.

Debouk *et al.* [13] addressed the problem of failure diagnosis in DES with decentralized information. A coordinated decentralized architecture was proposed that consists of two local sites communicating with a coordinator that is responsible for diagnosing the failures occurring in the system. They extend the notion of diagnosability, originally introduced in [40] for centralized systems, to the proposed coordinated decentralized architecture. In particular, they specify three protocols that realize the proposed architecture and analyze the diagnostic properties of these protocols.

Boel and van Schuppen [4] addressed the problem of synthesizing communication protocols and failure diagnosis algorithms for decentralized failure diagnosis of DES with costly communication between diagnosers. The costs on the communication channels may be described in terms of bits and complexity. The costs of communication and computation force the trade-off between the control objective of failure diagnosis and that of minimization of the costs of communication and computation. The main result of this paper is an algorithm for decentralized failure diagnosis of DES for the special case of only two diagnosers.

3.4 Diagnosis Using Petri Nets

Among the first pioneer works dealing with PNs, we recall the approach of Prock that in [31] proposed an on-line technique for fault detection that is based on monitoring the number of tokens residing into P-invariants: when the number of tokens inside P-invariants changes, then the error is detected.

Srinivasan and Jafari [42] employed time PNs to model a DES controller and backfiring transitions to determine whether a given state is invalid. Time PNs have also been considered by Ghazel *et al.* [17] to propose a monitoring approach for DES with unobservable events and to represent the "a priori" known behavior of the system, and track on-line its state to identify the events that occur.

Hadjicostis and Verghese [22] considered PN models introducing redundancy into the system and using P-invariants to allow the detection and isolation of faulty markings. Redundancy into a given PN was also exploited by Wu and Hadjicostis [47] to enable fault detection and identification using algebraic decoding techniques. Two types of faults are considered: place faults that corrupt the net marking, and transition faults that cause an incorrect update of the marking after event occurrence. Although this approach is general, the net marking has to

be periodically observable even if unobservable events occur. Analogously, Lefebvre and Delherm [27] investigated a procedure to determine the set of places that must be observed for the exact and immediate estimation of faults occurrence.

Note that all above mentioned papers assume that the marking of certain places may be observed. On the contrary other approaches, that we review in the rest of this subsection, are based on the assumption that no place is observable.

The three different approaches mentioned in the following [3,2,14] are characterized by the fact that they use on-line approaches to diagnosis. This may offer some advantages in terms of generality and adaptability, but may require heavy computations in real time.

Benveniste *et al.* [3] derived a net unfolding approach for designing an on-line asynchronous diagnoser. The state explosion is avoided but the computational complexity can be high due to the requirement of unfolding on-line PN structures.

Basile *et al.* [2] considered an on-line diagnoser that requires solving Integer Linear Programming (ILP) problems. Assuming that the fault transitions are not observable, the net marking is computed by the state equation and, if the marking has negative components, an unobservable sequence has occurred. The linear programming solution provides the sequence and detects the fault occurrences. Moreover, an off-line analysis of the PN structure may reduce the computational complexity of solving the ILP problem.

Dotoli *et al.* [14] proposed a diagnoser that works on-line as a means to avoid its redesign when the structure of the system changes. In particular, the diagnoser waits for an observable event and an algorithm decides whether the system behavior is normal or may exhibit some possible faults. To this aim, some ILP problems are defined and provide eventually the minimal sequences of unobservable transitions containing the faults that may have occurred. The proposed approach is a general technique since no assumption is imposed on the reachable state set that can be unlimited, and only few properties must be fulfilled by the structure of the net modeling the system fault behavior.

The state estimation approach founded on basis markings introduced by our group in [12] has also been extended to address diagnosis of Petri nets. In [8] the fundamental diagnosis approach was presented assuming that all observable transitions have a distinct label: the two notions of basis marking and justification allow one to characterize the set of markings that are consistent with the actual observation, and the set of unobservable transitions whose firing enables it. This approach applies to all net systems whose unobservable subnet is acyclic. If the net system is also bounded the proposed approach may be significantly simplified by moving the most burdensome part of the procedure off-line, thanks to the construction of a graph, called the *basis reachability graph*. In [6] the approach has been extended to the case of nets with observable transitions sharing the same label.

Ru and Hadjicostis [36] studied fault diagnosis in discrete event systems modeled by partially observed Petri nets, i.e., Petri nets equipped with sensors that allow observation of the number of tokens in some of the places and/or partial

observation of the firing of some of the transitions. Given an ordered sequence of observations from place and transition sensors, the goal is to calculate the belief (namely, the degree of confidence) regarding the occurrence of faults belonging to each type. This is done transforming a given partially observed PN into an equivalent labeled PN, i.e., a net with only transition sensors.

Finally, we mention the distributed approach by Genc and Lafortune [16] where the net is modular and local diagnosers performs the diagnosis of faults in each module. Subsequently, the local diagnosers recover the monolithic diagnosis information obtained when all the modules are combined into a single module that preserves the behavior of the underlying modular system. A communication system connects the different modules and updates the diagnosis information. Even if the approach does not avoid the state explosion problem, an improvement is obtained when the system can be modeled as a collection of PN modules coupled through common places.

3.5 Diagnosability Using Petri Nets

It should be noted that none of the above mentioned approaches for the diagnosis of PNs deal with *diagnosability*, namely none of them provide a procedure to determine a priori if a system is *diagnosable*, i.e., if it is possible to reconstruct the occurrence of fault events observing words of finite length.

The first contribution on diagnosability of PNs is due to Ushio *et al.* [44]. The main idea is to extend the diagnosability approaches for automata presented in [40,41] to deal with unbounded PNs, assuming that the set of places is partitioned into observable and unobservable places, while all transitions are unobservable. Starting from the net they build a diagnoser called *simple ω diagnoser* that gives sufficient conditions for diagnosability of unbounded PNs.

Chung [11] generalized the previous setting assuming that some of the transitions of the net are observable and showing that the additional information from observed transitions in general adds diagnosability to the analyzed system. Moreover starting from the diagnoser he proposed an automaton called *verifier* that allows a polynomial check of diagnosability for finite state models.

Wen and Jeng [45] proposed an approach to test diagnosability by checking the T-invariants of the nets. They used Ushio's diagnoser to prove that their method is correct, however they do not construct a diagnoser for the system to do diagnosis. Wen *et al.* [46] also presented an algorithm, based on a linear programming problem, of polynomial complexity in the number of nodes, to compute a sufficient diagnosability condition for DES modeled by PNs.

Jiroveanu and Boel [24] used a reduced automaton for testing diagnosability of labeled nets with unobservable transitions. The automaton generates the same language as the net reachability graph after projecting out all non-faulty unobservable transitions, and contains significantly less states than the reachability graph. The reduced automaton is efficiently constructed computing the minimal explanations of the fault and of the observable transitions.

Our group described a diagnosability test for the diagnosis approach founded on basis markings in [7]. The proposed approach applies to bounded nets with unobservable transitions and is based on the analysis of two graphs that depend on the structure of the net, including the faults model, and on the initial marking. The first graph is called *basis reachability diagnoser*, the second one is called *modified basis reachability graph*.

In [5] necessary and sufficient conditions for diagnosability of unbounded nets were given. The constructive procedure to test diagnosability is based on the analysis of the coverability graph of a particular net, called *verifier net*. To the best of our knowledge, this is the first available test that provides necessary and sufficient conditions for diagnosability of labeled unbounded Petri nets.

References

1. Barrett, G., Lafortune, S.: Decentralized supervisory control with communicating controllers. IEEE Trans. on Automatic Control 45(9), 1620–1638 (2000)
2. Basile, F., Chiacchio, P., De Tommasi, G.: An efficient approach for online diagnosis of discrete event systems. IEEE Trans. on Automatic Control 54(4), 748–759 (2008)
3. Benveniste, A., Fabre, E., Haar, S., Jard, C.: Diagnosis of asynchronous discrete event systems: A net unfolding approach. IEEE Trans. on Automatic Control 48(5), 714–727 (2003)
4. Boel, R.K., van Schuppen, J.H.: Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers. In: Proc. 6th Work. on Discrete Event Systems, Zaragoza, Spain, pp. 175–181 (October 2002)
5. Cabasino, M.P., Giua, A., Lafortune, S., Seatzu, C.: Diagnosability analysis of unbounded Petri nets. In: Proc. 48th IEEE Conf. on Decision and Control, Shanghai, China, pp. 1267–1272 (December 2009)
6. Cabasino, M.P., Giua, A., Pocci, M., Seatzu, C.: Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems. Control Engineering Practice (2010), doi:10.1016/j.conengprac.2010.12.010
7. Cabasino, M.P., Giua, A., Seatzu, C.: Diagnosability of bounded Petri nets. In: Proc. 48th IEEE Conf. on Decision and Control, Shanghai, China, pp. 1254–1260 (December 2009)
8. Cabasino, M.P., Giua, A., Seatzu, C.: Fault detection for discrete event systems using Petri nets with unobservable transitions. Automatica 46(9), 1531–1539 (2010)
9. Caines, P.E., Greiner, R., Wang, S.: Dynamical logic observers for finite automata. In: Proc. 27th IEEE Conf. on Decision and Control, Austin, TX, USA, pp. 226–233 (December 1988)
10. Caines, P.E., Wang, S.: Classical and logic based regulator design and its complexity for partially observed automata. In: Proc. 28th IEEE Conf. on Decision and Control, Tampa, FL, USA, pp. 132–137 (December 1989)
11. Chung, S.L.: Diagnosing PN-based models with partial observable transitions. International Journal of Computer Integrated Manufacturing 12 (2), 158–169 (2005)
12. Corona, D., Giua, A., Seatzu, C.: Marking estimation of Petri nets with silent transitions. IEEE Trans. on Automatic Control 52(9), 1695–1699 (2007)
13. Debouk, R., Lafortune, S., Teneketzis, D.: Coordinated decentralized protocols for failure diagnosis of discrete-event systems. Discrete Events Dynamical Systems 10(1), 33–86 (2000)

14. Dotoli, M., Fanti, M.P., Mangini, A.M.: Fault detection of discrete event systems using Petri nets and integer linear programming. In: Proc. of 17th IFAC World Congress, Seoul, Korea (July 2008)
15. Dubreil, J., Darondeau, P., Marchand, H.: Supervisory control for opacity. IEEE Trans. on Automatic Control 55(5), 1089–1100 (2010)
16. Genc, S., Lafortune, S.: Distributed diagnosis of place-bordered Petri nets. IEEE Trans. on Automation Science and Engineering 4(2), 206–219 (2007)
17. Ghazel, M., Togueni, A., Bigang, M.: A monitoring approach for discrete events systems based on a time Petri net model. In: Proc. of 16th IFAC World Congress, Prague, Czech Republic (July 2005)
18. Giua, A.: Petri net state estimators based on event observation. In: Proc. 36th IEEE Conf. on Decision and Control, San Diego, CA, USA, pp. 4086–4091 (December 1997)
19. Giua, A., Corona, D., Seatzu, C.: State estimation of λ-free labeled Petri nets with contact-free nondeterministic transitions. Discrete Events Dynamical Systems 15(1), 85–108 (2005)
20. Giua, A., Seatzu, C.: Observability of place/transition nets. IEEE Trans. on Automatic Control 49(9), 1424–1437 (2002)
21. Giua, A., Seatzu, C., Basile, F.: Observer based state-feedback control of timed Petri nets with deadlock recovery. IEEE Trans. on Automatic Control 49(1), 17–29 (2004)
22. Hadjicostis, C.N., Verghese, G.C.: Monitoring discrete event systems using Petri net embeddings. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 188–207. Springer, Heidelberg (1999)
23. Hashtrudi Zad, S., Kwong, R.H., Wonham, W.M.: Fault diagnosis in discrete-event systems: framework and model reduction. IEEE Trans. on Automatic Control 48(7), 1199–1212 (2003)
24. Jiroveanu, G., Boel, R.K.: The diagnosability of Petri net models using minimal explanations. IEEE Trans. on Automatic Control 55(7), 1663–1668 (2010)
25. Jiroveanu, G., Boel, R.K., Bordbar, B.: On-line monitoring of large Petri net models under partial observation. Discrete Events Dynamical Systems 18(3), 323–354 (2008)
26. Kumar, R., Garg, V., Markus, S.I.: Predicates and predicate transformers for supervisory control of discrete event dynamical systems. IEEE Trans. on Automatic Control 38(2), 232–247 (1993)
27. Lefebvre, D., Delherm, C.: Diagnosis of DES with Petri net models. IEEE Trans. on Automation Science and Engineering 4(1), 114–118 (2007)
28. Lin, F.: Diagnosability of discrete event systems and its applications. Discrete Event Dynamic Systems 4(2), 197–212 (1994)
29. Meda-Campaña, M.E., Ramírez-Treviño, A., Malo, A.: Identification in discrete event systems. In: Proc. IEEE Int. Conf. on Systems, Man and Cybernetics, San Diego, CA, USA, pp. 740–745 (October 1998)
30. Özveren, C.M., Willsky, A.S.: Observability of discrete event dynamic systems. IEEE Trans. on Automatic Control 35(7), 797–806 (1990)
31. Prock, J.: A new tecnique for fault detection using Petri nets. Automatica 27(2), 239–245 (1991)
32. Ramadge, P.J.: Observability of discrete-event systems. In: Proc. 25th IEEE Conf. on Decision and Control, Athens, Greece, pp. 1108–1112 (December 1986)

33. Ramírez-Treviño, A., Rivera-Rangel, I., López-Mellado, E.: Observer design for discrete event systems modeled by interpreted Petri nets. In: Proc. 2000 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA, USA, pp. 2871–2876 (April 2000)
34. Ricker, S.L., Caillaud, B.: Mind the gap: Expanding communication options in decentralized discrete-event control. In: Proc. 46th IEEE Conf. on Decision and Control, New Orleans, LA, USA, p. 5924 (December 2007)
35. Ru, Y., Hadjicostis, C.N.: Bounds on the number of markings consistent with label observations in Petri nets. IEEE Trans. on Automation Science and Engineering 6(2), 334–344 (2009)
36. Ru, Y., Hadjicostis, C.N.: Fault diagnosis in discrete event systems modeled by partially observed Petri nets. Discrete Events Dynamical Systems 19(4), 551–575 (2009)
37. Saboori, A., Hadjicostis, C.N.: Opacity-enforcing supervisory strategies for secure discrete event systems. In: Proc. 47th IEEE Conf. on Decision and Control, Cancun, Mexico, pp. 889–894 (December 2008)
38. Saboori, A., Hadjicostis, C.N.: Opacity verification in stochastic discrete event systems. In: Proc. 49th IEEE Conf. on Decision and Control, Atlanta, GA, USA, pp. 6759–6764 (December 2010)
39. Saboori, A., Hadjicostis, C.N.: Reduced-complexity verification for initial-state opacity in modular discrete event systems. In: Proc. 10th Work. on Discrete Event Systems, Berlin, Germany (August/September 2010)
40. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete-event systems. IEEE Trans. on Automatic Control 40(9), 1555–1575 (1995)
41. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Failure diagnosis using discrete-event models. IEEE Trans. on Control Systems Technology 4(2), 105–124 (1996)
42. Srinivasan, V.S., Jafari, M.A.: Fault detection and monitoring using time Petri nets. IEEE Trans. on Systems, Man and Cybernetics 23(4), 1155–1162 (1993)
43. Tripakis, S.: Undecidable problems of decentralized observation and control on regular languages. Information Processing Letters 90(1), 21–28 (2004)
44. Ushio, T., Onishi, L., Okuda, K.: Fault detection based on Petri net models with faulty behaviors. In: Proc. 1998 IEEE Int. Conf. on Systems, Man, and Cybernetics, San Diego, CA, USA, pp. 113–118 (October 1998)
45. Wen, Y., Jeng, M.: Diagnosability analysis based on T-invariants of Petri nets. In: Proc. 2005 IEEE Int. Conf. on Networking, Sensing and Control, Tucson, AZ, USA, pp. 371–376 (March 2005)
46. Wen, Y., Li, C., Jeng, M.: A polynomial algorithm for checking diagnosability of Petri nets. In: Proc. 2005 IEEE Int. Conf. on Systems, Man, and Cybernetics, pp. 2542–2547 (October 2005)
47. Wu, Y., Hadjicostis, C.N.: Algebraic approaches for fault identification in discrete-event systems. IEEE Trans. Robotics and Automation 50(12), 2048–2053 (2005)

Forward Analysis and Model Checking for Trace Bounded WSTS*

Pierre Chambart, Alain Finkel, and Sylvain Schmitz

LSV, ENS Cachan & CNRS, Cachan, France
{chambart,finkel,schmitz}@lsv.ens-cachan.fr

Abstract. We investigate a subclass of well-structured transition systems (WSTS), the *bounded*—in the sense of Ginsburg and Spanier (Trans. AMS 1964) —complete deterministic ones, which we claim provide an adequate basis for the study of forward analyses as developed by Finkel and Goubault-Larrecq (ICALP 2009). Indeed, we prove that, unlike other conditions considered previously for the termination of forward analysis, boundedness is decidable. Boundedness turns out to be a valuable restriction for WSTS verification, as we show that it further allows to decide all ω-regular properties on the set of infinite traces of the system.

1 Introduction

General Context. Forward analysis using acceleration [7] is established as one of the most efficient practical means, albeit in general without termination guarantee, to tackle safety problems in infinite state systems, e.g. in the tools TReX [5], Lash [34], or Fast [6]. Even in the context of well-structured transition systems (WSTS)—a unifying framework for infinite systems, that abstracts away from Petri net theory, and endowed with a generic backward coverability algorithm due to Abdulla et al. [1]—forward algorithms are commonly felt to be more efficient than backward procedures: e.g. for lossy channel systems [4], although the backward procedure always terminates, only the non-terminating forward procedure is implemented in the tool TReX [5].

Acceleration techniques rely on symbolic representations of sets of states to compute exactly the effect of repeatedly applying a finite sequence of transitions w, i.e. the effect of w^*. The forward analysis terminates if and only if a finite sequence $w_1^* \cdots w_n^*$ of such accelerations deriving the full reachability set can be found, resulting in the definition of the *post* flattable* [7] class of systems. Despite evidence that many classes of systems are flattable [35], whether a system is post*-flattable is undecidable for general systems [7].

The Well Structured Case. Finkel and Goubault-Larrecq [23, 24] have recently laid new theoretical foundations for the forward analysis of deterministic WSTS—where determinism is understood with respect to transition labels—,

* Work supported by ANR project AVeriSS (ANR-06-SETIN-001).

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 49–68, 2011.

by defining *complete* deterministic WSTS (abbreviated cd-WSTS) as a means of obtaining finite representations for downward closed sets of states, ∞-*effective* cd-WSTS as those for which the acceleration of certain sequences can effectively be computed, and by proposing a conceptual forward Clover procedure à la Karp and Miller for computing the full cover of a cd-WSTS—i.e. the downward closure of its set of reachable states. Similarly to post* flattable systems, their procedure terminates if and only if the cd-WSTS at hand is *cover flattable*, which is undecidable [24]. As we show in this paper, post* flattability is also undecidable for cd-WSTS, thus motivating the search for even stronger sufficient conditions for termination.

This Work. Our aim with this paper was to find a reasonable decidable sufficient condition for the termination of the Clover procedure. We have found one such condition in the work of Demri et al. [18] with *trace flattable* systems, which are maybe better defined as the systems with a *bounded* trace language in the sense of Ginsburg and Spanier [31]: a language $L \subseteq \Sigma^*$ is bounded if there exists $n \in \mathbb{N}$ and n words $w_1, \ldots, w_n$ in Σ^* such that $L \subseteq w_1^* \cdots w_n^*$. The regular expression $w_1^* \cdots w_n^*$ is called a *bounded expression* for L.

Trace boundedness implies post* and cover flattability, while generalizing flat systems, which have been considered by many authors in a variety of contexts [e.g. 9, 27, 15, 14, 25, 7, 11]. Moreover, Demri et al. show that it allows to decide liveness properties for a restricted class of counter systems. However, to the best of our knowledge, nothing was known regarding the decidability of boundedness itself, apart from the 1964 proof of decidability for context-free grammars [31] and the 1969 one for equal matrix grammars [39].

We characterize boundedness for cd-WSTS and provide as our main contribution a generic decision algorithm in Sec. 3. We employ techniques vastly different from the "homomorphisms and semilinear Parikh images" approach of Ginsburg and Spanier [31] and Siromoney [39], since we rely on the results of Finkel and Goubault-Larrecq [23, 24] to represent the effect of certain transfinite sequences of transitions. We further argue that both the class of systems (deterministic WSTS) and the property (boundedness) are in some sense optimal: we prove that boundedness becomes undecidable if we relax the determinism condition, and that the less restrictive property of post* flattability is not decidable on deterministic WSTS.

It might seem so far that the issues are simply shifted from the algorithmic part (termination of forward analyses) to transforming a flattable system into a bounded one (when possible). But trace boundedness delivers much more than just coverability via forward analysis (treated in Sec. 4.1): we show that all ω-regular word properties can be checked against the set of infinite traces of bounded ∞-effective cd-WSTS, resulting in the first known recursive class of WSTS with decidable liveness (Sec. 4.2). Liveness properties are in general undecidable in cd-WSTS [3, 36]: techniques for propositional linear-time temporal logic (LTL) model checking are not guaranteed to terminate [20, 2] or limited to subclasses, like Petri nets [21]. As a consequence of our result, action-based

```
1  // Performs n invocations of the rpc() function
2  // with at most P>=1 simultaneous concurrent calls
3  piped_multirpc (int n) {
4    int sent = n, recv = n; rendezvous rdv;
5    while (recv > 0)
6      if (sent > 0 && recv − sent < P) {
7        post(rdv, rpc); // asynchronous call
8        sent−−;
9      } else { // sent == 0 || recv − sent >= P
10       wait(rdv); // a rpc has returned
11       recv−−;
12     }
13 }
```

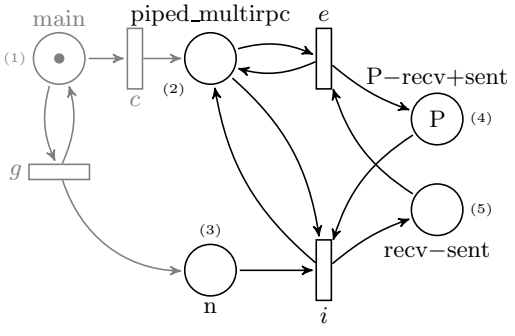


Fig. 1. A piped RPC client in C-like syntax, and its Petri net modelization

(aka transition-based) LTL model checking is decidable for bounded cd-WSTS, whereas state-based properties are undecidable for bounded cd-WSTS [16].

We finally consider the specific case of trace bounded Petri nets in Sec. 5, for which we show ExpSpace-completeness, and examine how boundedness helps for verification issues, since powerful branching-time logics become decidable [18]—they are indeed undecidable for general Petri nets [21].

This work results in an array of concrete classes of WSTS, including lossy channel systems [4], broadcast protocols [20], and Petri nets and their monotone extensions, such as reset/transfer Petri nets [19], for which boundedness is decidable and implies both computability of the full coverability set and decidability of liveness properties. Even for unbounded systems, it provides a new foundation for the heuristics currently employed by tools to help termination.

We refer the reader to a companion research report [13] for the missing proofs.

2 Background

2.1 A Running Example

We consider throughout this paper an example (see Fig. 1) inspired by the recent literature on *asynchronous* or *event-based* programming [33, 28], namely that of

a client performing n asynchronous remote procedure calls (corresponding to the post(r,rpc) statement on line 7), of which at most P can simultaneously be pending. Such piped—or windowed—clients are commonly employed to prevent server saturation.

The abstracted "producer/consumer" Petri net for this program (ignoring the grayed parts for now) has two transitions i and e modeling the **if** and **else** branches of lines 6 and 9 respectively. The deterministic choice between these two branches is here replaced by a nondeterministic one, where the program can choose the **else** branch and wait for some rpc call to return before the window of pending calls is exhausted. Observe that we can recover the original program behavior by further controlling the Petri net with the *bounded* regular language $i^P(ei)^*e^P$ (P is fixed), i.e. taking the intersection by synchronous product with a deterministic finite automaton for $i^P(ei)^*e^P$. This is an example of a trace bounded system.

Even without bounded control, the Petri net of Fig. 1 has a bounded, finite, language for each fixed initial n; however, for $P \geq 2$, if we expand it for parametric verification with the left grayed area to allow any n (or set $n = \omega$ as initial value to switch to server mode), then its language becomes unbounded. We will reuse this example in Sec. 3 when characterizing unboundedness in cd-WSTS. The full system is of course bounded when synchronized with a deterministic finite automaton for the language $g^*ci^P(ei)^*e^P$.

2.2 Definitions

Languages. Let Σ be a finite alphabet; we denote by Σ^* the set of finite sequences of elements from Σ, and by Σ^ω that of infinite sequences; $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote the empty sequence by ε. A language $L \subseteq \Sigma^*$ is *bounded* if there exists n and $w_1, \ldots, w_n$ in Σ^* s.t. $L \subseteq w_1^* \cdots w_n^*$. The left quotient of a language L_2 by a language L_1 over Σ is $L_1^{-1}L_2 = \{v \in \Sigma^* \mid \exists u \in L_1, uv \in L_2\}$.

We make regular use of the closure of bounded languages by finite union, intersection and concatenation, taking subsets, prefixes, suffixes, and factors, and of the following sufficient condition for the unboundedness of a language L [31, Lem. 5.3]: the existence of two words u and v in Σ^+, such that $uv \neq vu$ and each word in $\{u, v\}^*$ is a factor of some word in L.

Orderings. A *quasi order* $(S, \leq)$ is a set S endowed with a reflexive and transitive relation $\leq$, defining an equivalence relation $\leq \cap \leq^{-1}$. The *$\leq$-upward closure* $\uparrow C$ of a set $C \subseteq S$ is $\{s \in S \mid \exists c \in C, c \leq s\}$; its *$\leq$-downward closure* is $\downarrow C = \{s \in S \mid \exists c \in C, c \geq s\}$. A set C is *$\leq$-upward closed* (resp. $\leq$-downward closed) if $\uparrow C = C$ (resp. $\downarrow C = C$). A set B is a *basis* for an upward-closed set C (resp. downward-closed) if $\uparrow B = C$ (resp. $\downarrow B = C$). An upper bound $s \in S$ of a set A verifies $a \leq s$ for all a of A, while we denote its *least upper bound*, if it exists, by $\mathsf{lub}(A)$.

A *well quasi order* (wqo) is a quasi order $(S, \leq)$ such that for any infinite sequence $s_0 s_1 s_2 \cdots$ of S^ω there exist $i < j$ in $\mathbb{N}$ such that $s_i \leq s_j$. Equivalently, there does not exist any strictly descending chain $s_0 > s_1 > \cdots > s_i > \cdots$, and

any *antichain*, i.e. set of pairwise incomparable elements, is finite. In particular, the set of minimal elements of an upward-closed set C is finite modulo the natural equivalence, and is a basis for C. Pointwise comparison $\leq$ in $\mathbb{N}^k$, and scattered subword comparison $\preceq$ on finite sequences in Σ^* are well quasi orders by Dickson's and Higman's lemmata.

A *directed subset* $D \neq \emptyset$ of S is such that any pair of elements of D has an upper bound in D. A *directed complete partial order* (dcpo) is such that any directed subset has a least upper bound. A subset O of a dcpo is *open* if it is upward-closed and if, for any directed subset D such that $\mathsf{lub}(D)$ is in O, $D \cap O \neq \emptyset$. A partial function f on a dcpo is *partial continuous* if it is monotonic, $\mathsf{dom}f$ is open, and for any directed subset D of $\mathsf{dom}f$, $\mathsf{lub}(f(D)) = f(\mathsf{lub}(D))$. Partial continuous functions are used to describe the transitions of complete deterministic WSTS.

Transition Systems. A *labeled transition system* (LTS) $\mathcal{S} = \langle S, s_0, \Sigma, \rightarrow \rangle$ comprises a set S of states, an initial state $s_0 \in S$, a finite set of labels Σ, a transition relation $\rightarrow$ on S defined as the union of the relations $\xrightarrow{a} \subseteq S \times S$ for each a in Σ. The relations are extended to sequences in Σ^* by $s \xrightarrow{\varepsilon} s$ and $s \xrightarrow{aw} s''$ for a in Σ and w in Σ^* if there exists s' in S such that $s \xrightarrow{a} s'$ and $s' \xrightarrow{w} s''$. We write $\mathcal{S}(s)$ for the LTS $\langle S, s, \Sigma, \rightarrow \rangle$. A LTS is

- *bounded branching* if $\mathsf{Post}_{\mathcal{S}}(s) = \{s' \in S \mid s \rightarrow s'\}$ is bounded for all s in S,
- *deterministic* if $\xrightarrow{a}$ is a partial function for each a in Σ—we abuse notation in this case and identify u with the partial function $\xrightarrow{u}$ for u in Σ^*,
- *state bounded* if its *reachability set* $\mathsf{Post}^*_{\mathcal{S}}(s_0) = \{s \in S \mid s_0 \rightarrow^* s\}$ is finite,
- *trace bounded* if its *trace set* $T(\mathcal{S}) = \{w \in \Sigma^* \mid \exists s \in S, s_0 \xrightarrow{w} s\}$ is a bounded language in the sense of Ginsburg and Spanier.

A *well-structured transition system* (WSTS) [22, 1, 26] $\mathcal{S} = \langle S, s_0, \Sigma, \rightarrow, \leq, F \rangle$ is a labeled transition system endowed with a wqo $(S, \leq)$ and an $\leq$-upward closed set of final states F, such that $\rightarrow$ is *monotonic* wrt. $\leq$: for any s_1, s_2, s_3 in S and a in Σ, if $s_1 \leq s_2$ and $s_1 \xrightarrow{a} s_3$, then there exists $s_4 \geq s_3$ in S with $s_2 \xrightarrow{a} s_4$.

The *language* of a WSTS is defined as $L(\mathcal{S}) = \{w \in \Sigma^* \mid \exists s \in F, s_0 \xrightarrow{w} s\}$; see Geeraerts et al. [30] for a general study of such languages. In the context of Petri nets, $L(\mathcal{S})$ is also called the *covering* or *weak* language, and $T(\mathcal{S})$ the *prefix* language. Observe that a *deterministic finite-state automaton* (DFA) is a deterministic WSTS $\mathcal{A} = \langle Q, q_0, \Sigma, \delta, =, F \rangle$, where Q is finite (thus using $(Q, =)$ as underlying wqo).

Given two WSTS $\mathcal{S}_1 = \langle S_1, s_{0,1}, \Sigma, \rightarrow_1, \leq_1, F_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, s_{0,2}, \Sigma, \rightarrow_2, \leq_2, F_2 \rangle$, their *synchronous product* $\mathcal{S}_1 \times \mathcal{S}_2 = \langle S_1 \times S_2, (s_{0,1}, s_{0,2}), \Sigma, \rightarrow_\times, \leq_\times, F_1 \times F_2 \rangle$, where for all s_1, s_1' in S_1, s_2, s_2' in S_2, a in Σ, $(s_1, s_2) \xrightarrow{a}_\times (s_1', s_2')$ iff $s_1 \xrightarrow{a}_1 s_1'$ and $s_2 \xrightarrow{a}_2 s_2'$, and $(s_1, s_2) \leq_\times (s_1', s_2')$ iff $s_1 \leq_1 s_1'$ and $s_2 \leq_2 s_2'$, is again a WSTS, such that $L(\mathcal{S}_1 \times \mathcal{S}_2) = L(\mathcal{S}_1) \cap L(\mathcal{S}_2)$.

We often consider the case $F = S$ and omit F from the WSTS definition, as we are more interested in trace sets, which provide more evidence on the reachability sets.

Coverability. A WSTS is Pred-*effective* if $\rightarrow$ and $\leq$ are decidable, and a finite basis for $\uparrow\mathsf{Pred}_{\mathcal{S}}(\uparrow s, a) = \uparrow\{s' \in S \mid \exists s'' \in S,\ s' \xrightarrow{a} s'' \text{ and } s \leq s''\}$ can effectively be computed for all s in S and a in Σ [26]. The *cover set* of a WSTS is $\mathsf{Cover}_{\mathcal{S}}(s_0) = \downarrow\mathsf{Post}^*_{\mathcal{S}}(s_0)$, and it is decidable whether a given state s belongs to $\mathsf{Cover}_{\mathcal{S}}(s_0)$ for finite branching effective WSTS, thanks to a backward algorithm that checks whether s_0 belongs to $\uparrow\mathsf{Pred}^*_{\mathcal{S}}(\uparrow s) = \uparrow\{s' \in S \mid \exists s'' \in S, s' \rightarrow^* s'' \text{ and } s'' \geq s\}$ [1]. One can also decide the emptiness of the language of a WSTS, by checking whether s_0 belongs to $\uparrow\mathsf{Pred}^*_{\mathcal{S}}(F)$ [30].

Flattenings. Let $\mathcal{A}$ be a DFA with a bounded language. The synchronous product $\mathcal{S}'$ of $\mathcal{S}$ and $\mathcal{A}$ is a *flattening* of $\mathcal{S}$. Consider the projection π from $S \times Q$ to S defined by $\pi(s, q) = s$; then $\mathcal{S}$ is *post** *flattable* if there exists a flattening $\mathcal{S}'$ of $\mathcal{S}$ such that $\mathsf{Post}^*_{\mathcal{S}}(s_0) = \pi(\mathsf{Post}^*_{\mathcal{S}'}((s_0, q_0)))$. In the same way, it is *cover flattable* if $\mathsf{Cover}_{\mathcal{S}}(s_0) = \pi(\mathsf{Cover}_{\mathcal{S}'}((s_0, q_0)))$, and *trace flattable* if $T(\mathcal{S}) = T(\mathcal{S}')$. Remark that

1. trace flattability is equivalent to the boundedness of the trace set, and that
2. trace flattability implies post* flattability, which in turn implies cover flattability.

By a *flat* system we understand a system of form $\mathcal{S} \times \mathcal{A}$ with an explicit finite control $\mathcal{A}$ (i.e. a DFA) with bounded language. Whether such a system $\mathcal{S} \times \mathcal{A}$ is flat can be checked in PTIME [29], but, (1) if not, the system might still be trace flattable, and (2) not every system is endowed with an explicit finite state control, for instance Petri nets are not.

Complete WSTS. A deterministic WSTS $\langle S, s_0, \Sigma, \rightarrow, \leq\rangle$ is *complete* (a cd-WSTS) if $(S, \leq)$ is a continuous dcpo and each transition function a for a in Σ is partial continuous [23, 24]. The *lub-acceleration* u^ω of a partial continuous function u on S, u in Σ^+, is again a partial function on S defined by $\mathsf{dom}\, u^\omega = \{s \in \mathsf{dom}\, u \mid s \leq u(s)\}$ and $u^\omega(s) = \mathsf{lub}(\{u^n(s) \mid n \in \mathbb{N}\})$ for s in $\mathsf{dom}\, u^\omega$. A complete WSTS is ∞-*effective* if u^ω is computable for every u in Σ^+.

2.3 Working Hypotheses

Our decidability results rely on some effectiveness assumptions for a restricted class of WSTS: the complete deterministic ones. We discuss in this section the exact scope of these hypotheses. As an appetizer, notice that both boundedness and action-based ω-regular properties are only concerned with trace sets, hence one can more generally consider classes of WSTS for which a trace-equivalent complete deterministic system can effectively be found.

Completeness. Finkel and Goubault-Larrecq [24] define ω^2-*WSTS* as the class of systems that can be completed, and provide an extensive off-the-shelf algebra of datatypes (i.e. the finite disjoint sums, finite cartesian products, finite sequences, and finite multiset operations, with finite sets and $\mathbb{N}$ as base types) with their completions [23]. As they argue, all the concrete classes of deterministic WSTS considered in the literature are ω^2. Completed systems share their sets of finite

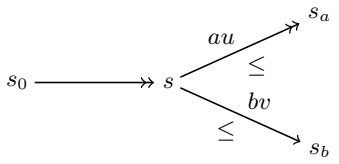


Fig. 2. An increasing fork witnesses unboundedness

and infinite traces with the original systems: the added limit states only influence transfinite sequences of transitions.

For instance, the whole class of *affine counter systems*, with affine transition functions of form $f(X) = AX + B$, with A a $k \times k$ matrix of non-negative integers and B a vector of k integers—encompassing reset/transfer Petri nets and broadcast protocols—can be completed to configurations in $(\mathbb{N} \cup \{\omega\})^k$. Similarly, functional lossy channel systems [23] can work on *products* [2, Coro. 6.5]. On both accounts, the completed functions are partial continuous.

Effectiveness. All the concrete classes of WSTS are Pred-effective, and we assume this property for all our systems from now on. It also turns out that ∞-effective systems abound, including once more (completed) affine counter systems [24] and functional lossy channel systems.

3 Deciding Trace Boundedness

We present in this section our algorithm for deciding trace boundedness (Sec. 3.1), which relies on a characterization using so-called *increasing forks*, and some udecidable cases when our working hypotheses are relaxed (Sec. 3.2).

The main theorem of this paper is:

Theorem 1. *Trace boundedness is decidable for ∞-effective cd-WSTS. If the trace set is bounded, then one can effectively find an adequate bounded expression $w_1^* \cdots w_n^*$ for it.*

3.1 Characterizing Unboundedness

Our construction relies on the existence of a witness of unboundedness, which can be found after a finite time in a cd-WSTS by exploring its states using accelerated sequences. We call this witness of unboundedness an *increasing fork*, as depicted in schematic form in Fig. 2. Let us first define accelerated runs and traces for complete WSTS, where lub-accelerations are employed.

Definition 1. *Let $\mathcal{S} = \langle S, s_0, \Sigma, \to, \leq, F \rangle$ be a cd-WSTS. An* accelerated run *is a finite sequence $\sigma = s_0 s_1 s_2 \cdots s_n$ in S^* such that for all $i \geq 0$, either there exists a in Σ such that*

$$s_i \xrightarrow{a} s_{i+1} \qquad \text{(single step)}$$

or there exists u in Σ^+ such that

$$s_i \xrightarrow{u^\omega} s_{i+1} \ . \qquad \text{(accelerated step)}$$

We denote the relation over S defined by such an accelerated run by $s_0 \twoheadrightarrow s_n$. The accelerated trace set $T_{\mathsf{acc}}(\mathcal{S})$ *of $\mathcal{S}$ is the set of sequences that label some accelerated run.*

We denote by $\Sigma^{<\omega^2}$ the set of sequences of (ordinal) length strictly smaller than ω^2; in particular $T_{\mathsf{acc}}(\mathcal{S}) \subseteq \Sigma^{<\omega^2}$.

Definition 2. *A cd-WSTS $\mathcal{S} = \langle S, s_0, \Sigma, \to, \leq \rangle$ has an* increasing fork *if there exist $a \neq b$ in Σ, u in $\Sigma^{<\omega^2}$, v in Σ^*, and s, $s_a \geq s$, $s_b \geq s$ in S such that $s_0 \twoheadrightarrow s$, $s \xrightarrow{au} s_a$, and $s \xrightarrow{bv} s_b$.*

Proposition 1. *A cd-WSTS has an unbounded trace set iff it has an increasing fork.*

An example. We first describe the intuition behind Prop. 1 before turning to the actual proof details. Consider the full Petri net $\mathcal{N}$ depicted in Fig. 1 with initial marking $(1, 0, 0, P, 0)$ for $P \geq 2$. As mentioned in Sec. 2, its trace set is unbounded, but the trace set of $\mathcal{N}$ with initial marking $(0, 1, n, P, 0)$ is bounded for each n, since it has a finite trace set. The unboundedness of $\mathcal{N}(1, 0, 0, P, 0)$ originates in its ability to reach each $(0, 1, n, P, 0)$ marking after a sequence of n transitions on g followed by a c transition.

Consider now transitions $(1, 0, 0, P, 0) \xrightarrow{g} (1, 0, 1, P, 0)$ and $(1, 0, 0, P, 0) \xrightarrow{c} (0, 1, 0, P, 0)$. The two systems $\mathcal{N}(1, 0, 1, P, 0)$ and $\mathcal{N}(0, 1, 0, P, 0)$ are respectively unbounded and bounded. In fact, for an unbounded language $L \subseteq \Sigma^*$, there always exist at least one a in Σ such that $a^{-1}L$ remains unbounded (see Lem. 3).

By induction, we can find words w of any length $|w| = n$ such that $w^{-1}T(\mathcal{S})$ is still unbounded: this is the case of g^n in our example. This process continues to the infinite, but in a WSTS we will eventually find two states $s_i \leq s_j$, met after $i < j$ steps respectively. Let $s_i \xrightarrow{u} s_j$; by monotonicity we can recognize u^* starting from s_i. In a cd-WSTS, there is a lub-accelerated state s with $s_i \xrightarrow{u^\omega} s$ that represents the effect of all these u transitions; here $(1, 0, 0, P, 0) \xrightarrow{g^\omega} (1, 0, \omega, P, 0)$. The interesting point is that our lub-acceleration finds the correct residual trace set: $T(\mathcal{N}(1, 0, \omega, P, 0)) = (g^*)^{-1}T(\mathcal{N}(1, 0, 0, P, 0))$.

Again, we can repeatedly remove accelerated strings from the prefixes of our trace set and keep it unbounded. However, due to the wqo, an infinite succession of lub-accelerations allows us to nest some loops after a finite number of steps. Still with the same example, we reach $(1, 0, \omega, P, 0) \xrightarrow{ci} (0, 1, \omega, P - 1, 1)$, and—thanks to the lub-acceleration—the source of unboundedness is now visible because both $(0, 1, \omega, P - 1, 1) \xrightarrow{ei} (0, 1, \omega, P - 1, 1)$ and $(0, 1, \omega, P - 1, 1) \xrightarrow{ieei} (0, 1, \omega, P - 1, 1)$ are increasing (this is our increasing fork). By monotonicity, $T(\mathcal{N}(0, 1, \omega, P - 1, 1))$ contains $\{ei, ieei\}^*$. Here continuity comes into play to show that these limit behaviors are reflected in the set of finite traces of the system: in our example, for each string u in $\{ei, ieei\}^*$, there exists a finite n in $\mathbb{N}$ such that $g^n ciu$ is an actual trace of $\mathcal{N}(1, 0, 0, P, 0)$.

An Increasing Fork Implies Unboundedness. The following lemma shows that, thanks to continuity, what happens in accelerated runs is mirrored in finite runs.

Lemma 1. *Let $\mathcal{S}$ be a cd-WSTS and $n \geq 0$. If*

$$w_n = v_{n+1} u_n^\omega v_n \cdots u_1^\omega v_1 \in T_{\mathsf{acc}}(\mathcal{S})$$

with the u_i in Σ^+ and the v_i in Σ^*, then there exist $k_1, \ldots, k_n$ in $\mathbb{N}$, such that

$$w'_n = v_{n+1} u_n^{k_n} v_n \cdots u_1^{k_1} v_1 \in T(\mathcal{S}) \ .$$

Proof. We proceed by induction on n. In the base case where $n = 0$, $w_0 = v_1$ belongs trivially to $T(\mathcal{S})$—this concludes the proof if we are considering words in $T(\mathcal{S})$. For the induction part, let s be a state such that

$$s_0 \xrightarrow{v_{n+1} u_n^\omega} s \xrightarrow{v_n u_{n-1}^\omega v_{n-1} \cdots u_1^\omega v_1} s_f \ ,$$

i.e. $w_{n-1} = v_n u_{n-1}^\omega v_{n-1} \cdots u_1^\omega v_1$ is in $T_{\mathsf{acc}}(\mathcal{S}(s))$. Therefore, using the induction hypothesis, we can find $k_1, \ldots, k_{n-1}$ in $\mathbb{N}$ such that

$$w'_{n-1} = v_n u_{n-1}^{k_{n-1}} v_{n-1} \cdots u_1^{k_1} v_1 \in T(\mathcal{S}(s)) \ .$$

Because $\mathcal{S}$ is complete, $\xrightarrow{w'_{n-1}}$ is a partial continuous function, hence with an open domain O. This domain O contains in particular s, which by definition of u_n^ω is the lub of the directed set $\{s' \mid \exists m \in \mathbb{N}, s_0 \xrightarrow{v_{n+1} u_n^m} s'\}$. By definition of an open set, there exists an element s' in $\{s' \mid \exists m \in \mathbb{N}, s_0 \xrightarrow{v_{n+1} u_n^m} s'\} \cap O$, i.e. there exists k_n in $\mathbb{N}$ s.t. $s_0 \xrightarrow{v_{n+1} u_n^{k_n}} s'$ and s' can fire the transition sequence w'_{n-1}. □

Continuity is crucial for the soundness of our procedure, as can be better understood by considering the example of the WSTS $\mathcal{S}' = \langle \mathbb{N} \uplus \{\omega\}, 0, \{a, b\}, \rightarrow, \leq \rangle$ with the transitions

$$\forall n \in \mathbb{N}, n \xrightarrow{a} n+1, \qquad \omega \xrightarrow{a} \omega, \qquad \omega \xrightarrow{b} \omega \ .$$

We obtain a bounded set of finite traces $T(\mathcal{S}'(0)) = a^*$, but reach the configuration ω through lub-accelerations, and then find an increasing fork with $T(\mathcal{S}'(\omega)) = \{a, b\}^*$, an unbounded language. Observe that $\mathbb{N}$ is a directed set with ω as lub, thus the domain of $\xrightarrow{b}$ should contain some elements of $\mathbb{N}$ in order to be open: $\mathcal{S}'$ is not a complete WSTS.

Lemma 2. *Let $\mathcal{S}$ be a cd-WSTS. If $\mathcal{S}$ has an increasing fork, then $T(\mathcal{S})$ is unbounded.*

Proof. Suppose that $\mathcal{S}$ has an increasing fork with the same notations as in Def. 2, and let w in $\Sigma^{<\omega^2}$ be such that $s_0 \xrightarrow{w} s$. By monotonicity, we can fire

from s the accelerated transitions of au and the transitions of bv in any order and any number of time, hence

$$w\{au, bv\}^* \subseteq T_{\mathsf{acc}}(\mathcal{S}) \ .$$

Suppose now that $T(\mathcal{S})$ is bounded, i.e. that there exists $w_1, \ldots, w_n$ such that $T(\mathcal{S}) \subseteq w_1^* \cdots w_n^*$. Then, there exists a DFA $\mathcal{A} = \langle Q, q_0, \Sigma, \delta, F\rangle$ such that $L(\mathcal{A}) = w_1^* \cdots w_n^*$ and thus $T(\mathcal{S}) \subseteq L(\mathcal{A})$. Set $N = |Q|+1$. We have in particular

$$w(bv)^N au(bv)^N au \cdots au(bv)^N \in T_{\mathsf{acc}}(\mathcal{S})$$

with N repetitions of the $(bv)^N$ factor. By Lem. 1, we can find some adequate finite sequences $w', u_1, \ldots, u_{N-1}$ in Σ^* such that

$$w'(bv)^N au_1(bv)^N au_2 \cdots au_{N-1}(bv)^N \in T(\mathcal{S}) \ .$$

Because $T(\mathcal{S}) \subseteq L(\mathcal{A})$, this word is also accepted by $\mathcal{A}$, and we can find an accepting run for it. Since $N = |Q| + 1$, for each of the N occurrences of the $(bv)^N$ factor, there exists a state q_i in Q such that $\delta(q_i, (bv)^{k_i}) = q_i$ for some $k_i > 0$. Thus the accepting run in $\mathcal{A}$ is of form

$$\begin{aligned}
&q_0 \xrightarrow{w'(bv)^{N-k_1-k'_1}} q_1 \xrightarrow{(bv)^{k_1}} q_1 \xrightarrow{(bv)^{k'_1} au_1 (bv)^{N-k_2-k'_2}} q_2,\\
&q_2 \xrightarrow{(bv)^{k_2}} q_2 \xrightarrow{(bv)^{k'_2} au_2 \cdots au_{N-1}(bv)^{N-k_N-k'_N}} q_N,\\
&q_N \xrightarrow{(bv)^{k_N}} q_N \xrightarrow{(bv)^{k'_N}} q_f \in F
\end{aligned}$$

for some integers $k'_i \geq 0$. Again, since $N = |Q| + 1$, there exist $1 \leq i < j \leq N$ such that $q_i = q_j$, hence

$$\delta(q_i, (bv)^{k'_i} au_i \cdots au_{j-1}(bv)^{N-k_j-k'_j}) = q_i \ .$$

This implies that $\{(bv)^{k_i}, (bv)^{k'_i} au_i \cdots au_{j-1}(bv)^{N-k_j-k'_j}\}^*$ is contained in the set of factors of $L(\mathcal{A})$ with

$$(bv)^{k_i+k'_i} au_i \cdots au_{j-1}(bv)^{N-k_j-k'_j} \neq (bv)^{k'_i} au_i \cdots au_{j-1}(bv)^{N-k_j-k'_j+k_i}$$

since $a \neq b$, thus $L(\mathcal{A})$ is an unbounded language [31, Lem. 5.3], a contradiction. □

Unboundedness Implies an Increasing Fork. We follow the arguments presented on the example of Fig. 1, and prove that an increasing fork can always be found in an unbounded cd-WSTS.

Lemma 3. *Let $L \subseteq \Sigma^*$ be an unbounded language. There exists a in Σ such that $a^{-1}L$ is also unbounded.*

Proof. Observe that $L = \bigcup_{a\in\Sigma} a \cdot (a^{-1}L)$. If every $a^{-1}L$ were bounded, since bounded languages are closed by finite union and concatenation, L would also be bounded. □

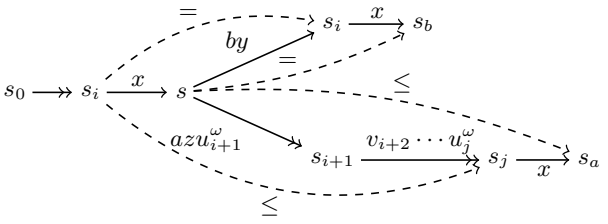


Fig. 3. The construction of an increasing fork in the proof of Lem. 5.

Definition 3. *Let be $L \subseteq \Sigma^*$ and $w \in \Sigma^+$. The* removal *of w from L is the language $\overline{w}L = (w^*)^{-1}L \backslash w\Sigma^*$.*

Lemma 4. *If a cd-WSTS $\mathcal{S}$ has an unbounded trace set $T(\mathcal{S})$ in Σ^*, and L is an unbounded subset of $T(\mathcal{S})$ then there are two words v in Σ^* and u in Σ^+ such that $vu^\omega \in T_{\mathsf{acc}}(\mathcal{S}), vu \in \mathsf{Pref}(L)$ and $\overline{u}(v^{-1}L)$ is also unbounded.*

Proof. By Lem. 3 we can find a sequence $(a_i)_{i>0} \in \Sigma^\omega$ such that for all n in $\mathbb{N}$, $(a_1 \cdots a_n)^{-1}L$ is unbounded. Let $(s_i)_{i\geq 0}$ be the corresponding sequence of configurations in S^ω, such that $s_i \xrightarrow{a_{i+1}} s_{i+1}$. Because $(S, \leq)$ is a wqo, there exist $i < j$ such that $s_i \leq s_j$. We set $v = a_1 \cdots a_i$ and $u = a_{i+1} \cdots a_j$, which gives us $v \cdot u^\omega \in T_{\mathsf{acc}}(\mathcal{S})$. Remark that $v^{-1}L$ is unbounded, and, since $u^*\overline{u}(v^{-1}L) = u^*(v^{-1}L)$, $\overline{u}(v^{-1}L)$ is unbounded too. □

Note that it is also possible to ask that $|vu| \geq n$ for any given n, which we do in the proof of the following lemma.

Lemma 5. *If a cd-WSTS has an unbounded trace set, then it has an increasing fork.*

Proof. We define simultaneously three infinite sequences, $(v_i, u_i)_{i>0}$ of pairs of words in $\Sigma^* \times \Sigma^+$, $(L_i)_{i\geq 0}$ of unbounded languages, and $(s_i)_{i\geq 0}$ of initial configurations: let $L_0 = T(\mathcal{S})$, and

- v_{i+1}, u_{i+1} are chosen using Lem. 4 such that $v_{i+1}u_{i+1}^\omega$ is in $T_{\mathsf{acc}}(\mathcal{S}(s_i))$, $v_{i+1}u_{i+1}$ is in $\mathsf{Pref}(L_i)$, $|v_{i+1} \cdot u_{i+1}| \geq |u_i|$ if $i > 0$, and $\overline{u_{i+1}}(v_{i+1}^{-1}L_i)$ is unbounded;
- $s_i \xrightarrow{v_{i+1}u_{i+1}^\omega} s_{i+1}$;
- $L_{i+1} = \overline{u_{i+1}}(v_{i+1}^{-1}L_i)$.

Since $\overline{u_{i+1}}(v_{i+1}^{-1}L_i) \subseteq T(\mathcal{S}(s_{i+1}))$, we can effectively iterate the construction by the last point above.

Due to the wqo, there exist $i < j$ such that $s_i \leq s_j$. By construction u_i is not a prefix of $v_{i+1}u_{i+1}$ and $|v_{i+1}u_{i+1}| \geq |u_i|$, so there exist $a \neq b$ in Σ and a longest common prefix x in Σ^* such that $u_i = xby$ and $v_{i+1}u_{i+1} = xaz$.

We exhibit an increasing fork by selecting s, s_a, s_b such that (see Fig. 3):

$$s_i \xrightarrow{x} s \qquad s \xrightarrow{azu_{i+1}^\omega v_{i+2}u_{i+2}^\omega \cdots v_j u_j^\omega x} s_a \qquad s \xrightarrow{byx} s_b\ .$$ □

From Increasing Forks to an Algorithm. To decide whether a cd-WSTS has an unbounded trace set, we do not attempt to find an increasing fork by an exhaustive search over all the accelerated sequences: indeed, the fact that the "b branch" of the increasing fork does not employ acceleration would prevent this search from terminating in the bounded case.

We look instead for a witness in the form of the sequences $(v_i, u_i)_{i>0}$ and $(s_i)_{i\geq 0}$ defined in the proof of Lem. 5. The proof itself shows that these sequences also characterize unboundedness. In order to obtain an algorithm, one merely has to note that, by monotonicity, choosing the shortest possible (v_i, u_i) at each step—i.e. accelerating as early as possible—is sufficient. Thus an exhaustive search either terminates upon finding $i < j$ with $s_i \leq s_j$ in the unbounded case, or terminates thanks to the wqo property in the bounded case. In the latter case, we end up with a finite tree from which the bounded expression can be extracted.

3.2 Undecidable Cases

Decidability disappears if we consider more general systems or a more general property: Trace boundedness becomes undecidable if we relax the determinism condition (which is required by our proofs when we use complete systems and accelerations), by considering the case of labeled reset Petri nets. We also show that post* flattability is undecidable in cd-WSTS, by a reduction from state boundedness in functional lossy channel systems.

Proposition 2. *Trace boundedness is undecidable for labeled (non deterministic) reset Petri nets.*

Proposition 3. *Post* flattability is undecidable for functional lossy channel systems.*

4 Verifying Bounded WSTS

As already mentioned in the introduction, liveness is generally undecidable for cd-WSTS. We show in this section that it becomes decidable for trace bounded systems obtained as the product of a cd-WSTS $\mathcal{S}$ with a deterministic Rabin automaton: we prove that it is decidable whether the language of ω-words of such a system is empty (Sec. 4.2) and apply it to the LTL model checking problem; but first we emphasize again the interest of boundedness for forward analysis techniques.

4.1 Forward Analysis

Recall from the introduction that a forward analysis of the set of reachable states in an infinite LTS typically relies on *acceleration techniques* [see e.g. 7] applied to loops w in Σ^*, provided one can effectively compute the effect of w^*. Computing the full reachability set (resp. coverability set for cd-WSTS) using a sequence $w_1^* \cdots w_n^*$ requires post* flattability (resp. cover flattability); however, as seen

with Prop. 3 (resp. [24, Prop. 6]), both these properties are already undecidable for cd-WSTS.

Trace bounded systems answer this issue since we can compute an appropriate finite sequence $w_1, \ldots, w_n$ and use it as acceleration sequence. Thus forward analysis techniques become complete for bounded systems. The Presburger accelerable counter systems of Demri et al. [18]—which include for instance reset/transfer Petri nets—are an example where, thanks to an appropriate representation for reachable states, the full reachability set is computable in the bounded case. In a more WSTS-centric setting, the forward Clover procedure of Finkel and Goubault-Larrecq for ∞-effective cd-WSTS terminates in the cover flattable case:

Corollary 1 (Finkel and Goubault-Larrecq [24]). *Let $\mathcal{S}$ be a trace bounded ∞-effective cd-WSTS. Then a finite representation of $\mathsf{Cover}_{\mathcal{S}}(s_0)$ can effectively be computed.*

Using the Cover set, one can answer state boundedness questions for WSTS. Furthermore, Cover sets and reachability sets coincide for lossy systems, and lossy channel systems in particular.

4.2 Deciding ω-Language Emptiness

ω-Regular Languages. Let us recall the Rabin acceptance condition for ω-words (indeed, our restriction to deterministic systems demands a stronger condition than the Büchi one). Let us set some notation for infinite words in a labeled transition system $\mathcal{S} = \langle S, s_0, \Sigma, \to \rangle$. A sequence of states σ in S^ω is an *infinite execution* for the infinite word $a_0 a_1 \cdots$ in Σ^ω if $\sigma = s_0 s_1 \cdots$ with $s_i \xrightarrow{a_i} s_{i+1}$ for all i. We denote by $T_\omega(\mathcal{S})$ the set of infinite words that have an execution. The *infinity set* of an infinite sequence $\sigma = s_0 s_1 \cdots$ in S^ω is the set of symbols that appear infinitely often in σ: $\mathsf{inf}(\sigma) = \{s \in S \mid |\{i \in \mathbb{N} \mid s_i = s\}| = \omega\}$.

Let $\mathcal{S} = \langle S, s_0, \Sigma, \to, \leq \rangle$ be a deterministic WSTS and $\mathcal{A} = \langle Q, q_0, \Sigma, \delta \rangle$ a DFA. A *Rabin acceptance condition* is a finite set of pairs $(E_i, F_i)_i$ of finite subsets of Q. An infinite word w in Σ^ω is accepted by $\mathcal{S} \times \mathcal{A}$ if its infinite execution σ over $(S \times Q)^\omega$ verifies $\bigvee_i (\mathsf{inf}(\sigma) \cap (S \times E_i) = \emptyset \wedge \mathsf{inf}(\sigma) \cap (S \times F_i) \neq \emptyset)$. The set of accepted infinite words is denoted by $L_\omega(\mathcal{S} \times \mathcal{A}, (E_i, F_i)_i)$. Thus an infinite run is accepting if, for some i, it goes only finitely often through the states of E_i, but infinitely often through the states of F_i.

We reduce the emptiness problem for $L_\omega(\mathcal{S} \times \mathcal{A}, (E_i, F_i)_i)$ to the boundedness problem for a finite set of cd-WSTS, which is decidable by Thm. 1. This does not hold for nondeterministic systems, since any system can be turned into a bounded one by simply relabeling every transition with a single letter a.

Theorem 2. *Let $\mathcal{S}$ be an ∞-effective cd-WSTS, $\mathcal{A}$ a DFA, and $(E_i, F_i)_i$ a Rabin condition. If $\mathcal{S} \times \mathcal{A}$ is trace bounded, then it is decidable whether $L_\omega(\mathcal{S} \times \mathcal{A}, (E_i, F_i)_i)$ is empty.*

Proof. Set $\mathcal{S} = \langle S, s_0, \Sigma, \to, \leq \rangle$ and $\mathcal{A} = \langle Q, q_0, \Sigma, \delta \rangle$.

We first construct one cd-WSTS $\mathcal{S}_{i,1}$ for each condition (E_i, F_i) by adding to Σ a fresh symbol e_i, to $S \times Q$ the pairs (s, q_i) where s is in S and q_i is a fresh state for each q in E_i, and replace in $\rightarrow$ each transition $(s,q) \xrightarrow{a} (s',q')$ of $\mathcal{S} \times \mathcal{A}$ with q in E_i by two transitions $(s,q) \xrightarrow{e_i} (s,q_i) \xrightarrow{a} (s',q')$. Thus we meet in $\mathcal{S}_i$ an e_i marker each time we visit some state in E_i.

Claim. Each $\mathcal{S}_{i,1}$ is a bounded cd-WSTS.

Proof of the claim. Observe that any trace of $\mathcal{S}_{i,1}$ is the image of a trace of $\mathcal{S} \times \mathcal{A}$ by a *generalized sequential machine* (GSM) $\mathcal{T}_i = \langle Q, q_0, \Sigma, \Sigma, \delta, \gamma \rangle$ constructed from $\mathcal{A} = \langle Q, q_0, \Sigma, \delta \rangle$ with the same set of states and the same transitions, and by setting the output function γ from $Q \times \Sigma$ to Σ^* to be

$$\begin{aligned} (q,a) &\mapsto e_i a && \text{if } q \in E_i \\ (q,a) &\mapsto a && \text{otherwise.} \end{aligned}$$

Since bounded languages are closed under GSM mappings [31, Coro. on p. 348] and $\mathcal{S} \times \mathcal{A}$ is bounded, we know that $\mathcal{S}_{i,1}$ is bounded. □

In a second phase, we add a new symbol f_i and the elementary loops $(s,q) \xrightarrow{f_i} (s,q)$ for each (s,q) in $S \times F_i$ to obtain a system $\mathcal{S}_{i,2}$. Any run that visits some state in F_i has therefore the opportunity to loop on f_i^*.

In $\mathcal{S} \times \mathcal{A}$, visiting F_i infinitely often implies that we can find two configurations $(s,q) \leq (s',q)$ with q in F_i. In $\mathcal{S}_{i,2}$, we can thus recognize any sequence in $\{f_i, w\}^*$, where $(s,q) \xrightarrow{w} (s',q)$, from (s',q): $\mathcal{S}_{i,2}$ is not bounded.

Claim. Each $\mathcal{S}_{i,2}$ is a cd-WSTS, and is unbounded iff there exists a run σ in $\mathcal{S} \times \mathcal{A}$ with $\mathsf{inf}(\sigma) \cap (S \times F_i) \neq \emptyset$.

Proof of the claim. If there exists a run σ in $\mathcal{S} \times \mathcal{A}$ with $\mathsf{inf}(\sigma) \cap (S \times F_i) \neq \emptyset$, then we can consider the infinite sequence of visited states in $S \times F_i$ along σ. Since $\leq$ is a well quasi ordering on $S \times Q$, there exist two steps (s,q) and later (s',q') in this sequence with $(s,q) \leq (s',q')$. Observe that the same execution σ, modulo the transitions introduced in $\mathcal{S}_{i,1}$, is also possible in $\mathcal{S}_{i,2}$. Denote by w in Σ^* the sequence of transitions between these two steps, i.e. $(s,q) \xrightarrow{w} (s',q')$. By monotonicity of the transition relation of $\mathcal{S}_{i,2}$, we can recognize any sequence in $\{f_i, w\}^*$ from (q',s'). Thus $\mathcal{S}_{i,2}$ is not bounded.

Conversely, suppose that $\mathcal{S}_{i,2}$ is not bounded. By Lem. 5, it has an increasing fork with $(s_0,q_0) \xrightarrow{w} (s,q) \xrightarrow{au} (s_a,q)$ and $(s,q) \xrightarrow{bv} (s_b,q)$, $s_a \geq s$, $s_b \geq s$, $a \neq b$ in $\Sigma \uplus \{e_i, f_i\}$, u, w in $(\Sigma \uplus \{e_i, f_i\})^{<\omega^2}$, and v in $(\Sigma \uplus \{e_i, f_i\})^*$.

Observe that if f_i only appears in the initial segment labeled by w, then a similar fork could be found in $\mathcal{S}_{i,1}$, since (s,q) would also be accessible. Thus, by Lem. 2, $\mathcal{S}_{i,1}$ would not be bounded. Therefore f_i appears in au or bv, and thus the corresponding runs for au or bv visit some state in F_i. But then, by monotonicity, we can construct a run that visits a state in F_i infinitely often. □

In the last, third step, we construct the synchronous product $\mathcal{S}_{i,3} = \mathcal{S}_{i,2} \times \mathcal{A}_i$, where $\mathcal{A}_i$ is a DFA for the language $(\Sigma \uplus \{e_i\})^* f_i (\Sigma \uplus \{f_i\})^*$ (where $\uplus$ denotes a

disjoint union). This ensures that any run of $\mathcal{S}_{i,3}$ that goes through at least one f_i cannot go through e_i any longer, hence it visits the states in E_i only finitely many often. Since a run can always choose not to go through a f_i loop, the previous claim still holds. Therefore each $\mathcal{S}_{i,3}$ is a cd-WSTS, is unbounded iff there exists a run σ in $\mathcal{S} \times \mathcal{A}$ with $\mathsf{inf}(\sigma) \cap (S \times E_i) = \emptyset$ and $\mathsf{inf}(\sigma) \cap (S \times F_i) \neq \emptyset$, and we can apply Thm. 1. □

Model Checking LTL Formulæ. By standard automata-theoretic arguments [41, 38], one can convert any linear-time temporal logic (LTL) formula φ over a finite set AP of atomic propositions, representing transition predicates, into a deterministic Rabin automaton $\mathcal{A}_{\neg\varphi}$ that recognizes exactly the runs over $\Sigma = 2^{\mathrm{AP}}$ that model $\neg\varphi$. The synchronized product of $\mathcal{A}_{\neg\varphi}$ with a complete, deterministic, ∞-effective, and trace bounded WSTS $\mathcal{S}$ is again trace bounded, and such that $L_\omega(\mathcal{S} \times \mathcal{A}, (E_i, F_i)_i) = T_\omega(\mathcal{S}) \cap L_\omega(\mathcal{A}, (E_i, F_i)_i)$. Thm. 2 entails that we can decide whether this language is empty, and whether all the infinite traces of $\mathcal{S}$ verify φ, noted $\mathcal{S} \models \varphi$. This reduction also works for LTL extensions that remain ω-regular.

Corollary 2. *Let $\mathcal{S} = \langle S, s_0, 2^{\mathrm{AP}}, \to, \leq \rangle$ be an ∞-effective trace bounded cd-WSTS, and φ a LTL formula on the set* AP *of atomic propositions. It is decidable whether $\mathcal{S} \models \varphi$.*

An alternative application of Thm. 2 is, rather than relying on the boundedness of $\mathcal{S}$, to ensure that $\mathcal{A}_{\neg\varphi}$ is bounded. To this end, one can adapt for instance the flat counter logic of Comon and Cortier [14].

Definition 4. *A LTL formula on a set* AP *of atomic propositions is* co-flat *if it is of form $\neg\varphi$, where φ follows the abstract syntax, where a stands for a letter in 2^{AP}:*

$$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \alpha \mathsf{U} \varphi \mid \mathsf{G}\alpha \qquad \text{(flat formulæ)}$$

$$\alpha ::= \bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \neg p \,. \qquad \text{(alphabetic formulæ)}$$

One can easily check that flat formulæ define languages of infinite words with bounded sets of finite prefixes, and we obtain:

Corollary 3. *Let $\mathcal{S} = \langle S, s_0, 2^{\mathrm{AP}}, \to, \leq \rangle$ be an ∞-effective cd-WSTS, and φ a co-flat LTL formula on the set* AP *of atomic propositions. It is decidable whether $\mathcal{S} \models \varphi$.*

Note that this particular application of Thm. 2 still relies on Thm. 1, since we reduce to the decidability of trace boundedness. Extensions of Coro. 3 to less restrictive LTL fragments are possible, but lead to rather unnatural conditions on the syntax of the restricted logic.

Beyond ω-Regular Properties. Not all properties are decidable for bounded cd-WSTS, as seen with the following theorem on affine counter systems. Since these

systems are otherwise completable, deterministic, and ∞-effective, action-based properties are decidable for them using Thm. 2, but we infer that state-based properties are undecidable for bounded ∞-effective cd-WSTS.

Theorem 3 (Cortier [16]). *Reachability is undecidable for trace bounded affine counter systems.*

5 The Petri Net Case

The interest of trace boundedness for Petri nets might not be as immediate as for more general classes of WSTS, since, for Petri nets, forward analyses always terminate (using the Karp and Miller algorithm [32]) and action-based LTL is decidable [21]. We will see in Sec. 5.2 that trace bounded Petri nets form an interesting class of systems, for which both the reachability and trace sets can be effectively described using Presburger arithmetic, and model checking for very powerful branching-time logics becomes decidable. But first we consider the complexity of trace boundedness for Petri nets, which we show to be ExpSpace-complete (Sec. 5.1).

5.1 Complexity

Well-structured transition systems are a highly abstract class of systems, for which no complexity upper bounds can be given in general. Nevertheless, it is possible to provide precise bounds for several concrete classes of WSTS, which we do here for Petri nets.[1]

Let us first recall that a marked *Petri net* is a tuple $\mathcal{N} = \langle P, \Theta, f, m_0 \rangle$ where P and Θ are finite sets of places and transitions, f a flow function from $(P \times \Theta) \cup (\Theta \times P)$ to $\mathbb{N}$, and m_0 an initial marking in $\mathbb{N}^P$. The set of *markings* $\mathbb{N}^P$ is ordered component-wise by $m \leq m'$ iff $\forall p \in P,\ m(p) \leq m'(p)$, and has the zero vector $\bar{0}$ as least element, such that $\forall p \in P,\ \bar{0}(p) = 0$. A transition $t \in \Theta$ can be fired in a marking m if $f(p,t) \geq m(p)$ for all $p \in P$, and reaches a new marking m' defined by $m'(p) = m(p) - f(p,t) + f(t,p)$ for all $p \in P$. A *labeled* Petri net (without ε labels) further associates a labeling letter-to-letter homomorphism $\sigma : \Theta \to \Sigma$, and can be seen as a WSTS $\langle \mathbb{N}^P, m_0, \Sigma, \to, \leq \rangle$ where $m \xrightarrow{\sigma(t)} m'$ if the transition t can be fired in m and reaches m'. An important class of Petri nets is defined by setting $\Sigma = \Theta$ and $\sigma = \mathrm{id}_\Theta$, thereby obtaining the so-called *free labeled* Petri nets—which define deterministic WSTS.

Upper Bound. We rely for the upper complexity bound on a characterization of trace unbounded Petri nets using their *coverability graph* [see e.g. 40]. Like many other Petri net properties, trace boundedness can be witnessed on this finite graph, and using techniques inspired by Rackoff's work [37], can be tested for in exponential space.

[1] See the companion research report [13] for the more involved cases of affine counter systems and functional lossy channel systems.

Proposition 4. *Trace boundedness of a labeled Petri net can be decided in exponential space.*

Proof Sketch. Given a labeled Petri net $\mathcal{N} = \langle P, \Theta, f, m_0, \sigma \rangle$, let us show that $T(\mathcal{N})$ is unbounded iff the coverability graph contains a node n with two circuits over transition sequences u and v in Θ^*, s.t. $\sigma(u)\sigma(v) \neq \sigma(v)\sigma(u)$.

Since the coverability graph is finite, we can see it as a finite automaton with prefix-closed language $C(\mathcal{N})$ with $T(\mathcal{N}) \subseteq C(\mathcal{N})$. The previous "two circuits" condition is equivalent to $C(\mathcal{N})$ being unbounded, hence its negation entails the boundedness of $C(\mathcal{N})$ and thus that of $T(\mathcal{N})$. Conversely, for any sequence w in $\{u, v\}^*$, there exists a finite transition sequence x_w s.t. $x_w w$ can be fired from m_0 [40, Thm. 1.(e)]. Thus by [31, Lem. 5.3], $T(\mathcal{N})$ is unbounded.

To conclude, note that we can choose n, u, and v such that $\sigma(u) = au'$ and $\sigma(v) = bv'$ with $a \neq b$ in Σ. Looking for such a pattern can be performed in exponential space using the techniques described by Demri [17], Blockelet and Schmitz [8]. □

Lower Bounds. Let us first match the above ExpSpace upper bound: we extend the ExpSpace hardness result of Lipton [12] for the Petri net coverability problem to the trace boundedness problem.

Proposition 5. *Deciding the trace boundedness of a free labeled Petri net is* ExpSpace*-hard.*

We also derive a non primitive recursive lower bound on the computation of the words $w_1, \ldots, w_n$. Indeed, the size of a covering tree can be non primitive recursive compared to the size of the Petri net [12, who attribute the idea to Hack]. Using the same insight, we demonstrate that the words $w_1, \ldots, w_n$ themselves can be of non primitive recursive size. This complexity is thus inherent to the computation of the w_i's.

Proposition 6. *There exists a free labeled Petri net $\mathcal{N}$ with a bounded trace set $T(\mathcal{N})$ but such that for any words $w_1, \ldots, w_n$, if $T(\mathcal{N}) \subseteq w_1^* \cdots w_n^*$, then the size $\sum_{i=1}^{n} |w_i|$ is not primitive recursive in the size of $\mathcal{N}$.*

5.2 Verifying Bounded Free Labeled Petri Nets

Semilinear Reachability Sets. Petri nets form a class of counter systems with *finite monoid* [25], and thus for which the effect of an iterated transition sequence w_i^* can be expressed in Presburger arithmetic: Petri nets are *Presburger accelerable* in the sense of Demri et al. [18]. In the case of a trace bounded Petri net, any reachable marking can be obtained as the result of a finite sequence of accelerations $w_1^* \cdots w_n^*$: the set of reachable markings is thus semilinear and effectively computable. This allows for instance to compare the reachability sets of two trace bounded Petri nets for inclusion (aka Post^* inclusion, undecidable for general Petri nets).

Table 1. Some decidability results for selected classes of cd-WSTS—Petri nets (PN), affine counter systems (ACS), and functional lossy channel systems (LCS)—in the trace unbounded and trace bounded cases

	PN	Bounded PN	ACS	Bounded ACS	LCS	Bounded LCS
Reachability	Yes	Yes	No	No	Yes	Yes
Post^* inclusion	No	Yes	No	No	No	Yes
Liveness	Yes	Yes	No	Yes	No	Yes

Semilinear Bounded Languages. The same reasoning allows to precisely characterize the language of a trace bounded Petri net: build a DFA with t transitions for the bounded expression $w_1^* \cdots w_n^*$ and synchronize it with the Petri net, using t extra places $p_1, \ldots, p_t$ incremented whenever the corresponding transition is fired. Projecting the semilinear reachability set on those t places shows that the language of a trace bounded Petri net can be described by an effectively constructible 1-dimensional CQDD [10], i.e. a DFA with bounded language constrained by a Presburger formula on the transition counts—CQDDs are used for model checking FIFO systems and they enjoy useful closure properties.

FOPCTL*(PrA) Model Checking. Finally, since Petri nets are Presburger acceleratable, model checking for the temporal logic FOPCTL*(PrA)—CTL* with past modalities, first-order quantification, and Presburger formulæ instead of atomic propositions—is decidable in the trace bounded case. This is an extremely powerful logic (for instance, both marking reachability and reachability set inclusion can be reduced to FOPCTL*(PrA) model-checking), which is undecidable for general Petri nets [21].

Observe finally that the properties of trace bounded Petri nets exhibited in this subsection immediately generalize to trace bounded Presburger acceleratable counter systems.

6 Boundedness Is Not a Weakness

To paraphrase the title *Flatness is not a Weakness* [14], boundedness is a powerful property for the analysis of systems, as demonstrated with the termination of forward analyses and the decidability of ω-regular properties for bounded WSTS (see also Table 1)—and is implied by flatness. Most prominently, boundedness has the considerable virtue of being decidable for a large class of systems, the ∞-effective complete deterministic WSTS.

One might fear boundedness is too strong a property to be of any practical use. For instance, commutations, as created by concurrent transitions, often result in unboundedness. However, bear in mind that the same issues more broadly affect all forward analysis techniques, and have been alleviated in tools through various heuristics. Boundedness offers a new insight into why such heuristics work, and can be used as a theoretical foundation for their principled development; we illustrate this point in the companion research report [13] where we introduce boundedness modulo a partial commutation relation.

References

1. Abdulla, P.A., Čerans, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Inform. and Comput. 160, 109–127 (2000)
2. Abdulla, P.A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B.: Using forward reachability analysis for verification of lossy channel systems. Form. Methods in Syst. Des. 25, 39–65 (2004)
3. Abdulla, P.A., Jonsson, B.: Undecidable verification problems for programs with unreliable channels. Inform. and Comput. 130, 71–90 (1996)
4. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inform. and Comput. 127, 91–101 (1996)
5. Annichini, A., Bouajjani, A., Sighireanu, M.: TREX: A tool for reachability analysis of complex systems. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 368–372. Springer, Heidelberg (2001)
6. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: Fast: acceleration from theory to practice. Int. J. Softw. Tools Technol. Transfer 10, 401–424 (2008)
7. Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 474–488. Springer, Heidelberg (2005)
8. Blockelet, M., Schmitz, S.: Model checking coverability graphs of vector addition systems (2011) (in preparation)
9. Boigelot, B., Wolper, P.: Symbolic verification with periodic sets. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 55–67. Springer, Heidelberg (1994)
10. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theor. Comput. Sci. 221, 211–250 (1999)
11. Bozga, M., Iosif, R., Lakhnech, Y.: Flat parametric counter automata. Fund. Inform. 91, 275–303 (2009)
12. Cardoza, E., Lipton, R.J., Meyer, A.R.: Exponential space complete problems for Petri nets and commutative semigroups. In: Proc. STOC 1976, pp. 50–54. ACM Press, New York (1976)
13. Chambart, P., Finkel, A., Schmitz, S.: Forward analysis and model checking for trace bounded WSTS. Research report, LSV (2010), `http://arxiv.org/abs/1004.2802` (cs.LO)
14. Comon, H., Cortier, V.: Flatness is not a weakness. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 262–276. Springer, Heidelberg (2000)
15. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and Presburger arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
16. Cortier, V.: About the decision of reachability for register machines. Theor. Inform. Appl. 36, 341–358 (2002)
17. Demri, S.: On Selective Unboundedness of VASS. In: Proc. INFINITY 2010. Elec. Proc. in Theor. Comput. Sci., vol. 39, pp. 1–15 (2010)
18. Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Model-checking CTL* over flat Presburger counter systems. J. Appl. Non-Classical Log. 20, 313–344 (2011)
19. Dufourd, C., Jančar, P., Schnoebelen, P.: Boundedness of reset P/T nets. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 301–310. Springer, Heidelberg (1999)
20. Emerson, E.A., Namjoshi, K.S.: On model checking for non-deterministic infinite-state systems. In: Proc. LICS 1998, pp. 70–80. IEEE, Los Alamitos (1998)

21. Esparza, J.: Decidability of model checking for infinite-state concurrent systems. Acta Inf. 34, 85–107 (1997)
22. Finkel, A.: Reduction and covering of infinite reachability trees. Inform. and Comput. 89, 144–179 (1990)
23. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part I: Completions. In: Proc. STACS 2009. LIPIcs, vol. 3, pp. 433–444. LZI (2009)
24. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part II: Complete WSTS. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 188–199. Springer, Heidelberg (2009)
25. Finkel, A., Leroux, J.: How to compose presburger-accelerations: Applications to broadcast protocols. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002)
26. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. 256, 63–92 (2001)
27. Fribourg, L., Olsén, H.: Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 213–227. Springer, Heidelberg (1997)
28. Ganty, P., Majumdar, R., Rybalchenko, A.: Verifying liveness for asynchronous programs. In: Proc. POPL 2009, pp. 102–113. ACM Press, New York (2009)
29. Gawrychowski, P., Krieger, D., Rampersad, N., Shallit, J.: Finding the growth rate of a regular or context-free language in polynomial time. Int. J. Fund. Comput. Sci. 21, 597–618 (2010)
30. Geeraerts, G., Raskin, J., Begin, L.V.: Well-structured languages. Acta Inf. 44, 249–288 (2007)
31. Ginsburg, S., Spanier, E.H.: Bounded Algol-like languages. T. Amer. Math. Soc. 113, 333–368 (1964)
32. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. 3, 147–195 (1969)
33. Krohn, M., Kohler, E., Kaashoek, M.F.: Events can make sense. In: Proc. USENIX 2007, pp. 87–100 (2007)
34. The Liège automata-based symbolic handler (Lash), `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`
35. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)
36. Mayr, R.: Undecidable problems in unreliable computations. Theor. Comput. Sci. 297, 337–354 (2003)
37. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. 6, 223–231 (1978)
38. Safra, S.: On the complexity of ω-automata. In: Proc. FOCS 1988, pp. 319–327. IEEE Computer Society Press, Los Alamitos (1988)
39. Siromoney, R.: A characterization of semilinear sets. Proc. Amer. Math. Soc. 21, 689–694 (1969)
40. Valk, R., Vidal-Naquet, G.: Petri nets and regular languages. J. Comput. Syst. Sci. 23, 299–325 (1981)
41. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. LICS 1986, pp. 332–344 (1986)

Minimal Coverability Set for Petri Nets: Karp and Miller Algorithm with Pruning*

Pierre-Alain Reynier[1] and Frédéric Servais[2]

[1] LIF, Université Aix-Marseille & CNRS, France
[2] Department of Computer & Decision Engineering (CoDE), ULB, Belgium

Abstract. This paper presents the Monotone-Pruning algorithm (MP) for computing the minimal coverability set of Petri nets. The original Karp and Miller algorithm (K&M) unfolds the reachability graph of a Petri net and uses acceleration on branches to ensure termination. The MP algorithm improves the K&M algorithm by adding pruning between branches of the K&M tree. This idea was first introduced in the Minimal Coverability Tree algorithm (MCT), however it was recently shown to be incomplete. The MP algorithm can be viewed as the MCT algorithm with a slightly more aggressive pruning strategy which ensures completeness. Experimental results show that this algorithm is a strong improvement over the K&M algorithm.

1 Introduction

Petri nets form an important formalism for the description and analysis of concurrent systems. While the state space of a Petri net may be infinite, many verification problems are decidable. The minimal coverability set (MCS) [2] is a finite representation of a well-chosen over-approximation of the set of reachable markings. As proved in [2], it can be used to decide several important problems. Among them we mention the *coverability* problem to which many safety problems can be reduced (is it possible to reach a marking dominating a given one?); the *boundedness* problem (is the set of reachable markings finite?); the *place boundedness* problem (given a place p, is it possible to bound the number of tokens in p in any reachable marking?); the *semi-liveness* problem (is there a reachable marking in which a given transition is enabled?). Finally, the *regularity problem* asks whether the set of reachable markings is regular.

Karp and Miller (K&M) introduced an algorithm for computing the MCS [8]. This algorithm builds a finite tree representation of the (potentially infinite) unfolding of the reachability graph of the given Petri net. It uses acceleration techniques to collapse branches of the tree and ensure termination. By taking advantage of the fact that Petri nets are strictly monotonic transition systems,

* Work partly supported by the French projects ECSPER (ANR-09-JCJC-0069) and DOTS (ANR-06-SETI-003), by the PAI program Moves funded by the Federal Belgian Government, by the European project QUASIMODO (FP7-ICT-STREP-214755), and by the ESF project GASICS.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 69–88, 2011.

the acceleration essentially computes the limit of repeatedly firing a sequence of transitions. The MCS can be extracted from the K&M tree. The K&M Algorithm thus constitutes a key tool for Petri nets, and has been extended to other classes of well-structured transition systems [1].

However, the K&M Algorithm is not efficient and in real-world examples it often does not terminate in reasonable time. One reason is that in many cases it will compute several times a same subtree; two nodes labelled with the same marking will produce the same subtree, K&M will compute both. This observation lead to the Minimal Coverability Tree (MCT) algorithm [2]. This algorithm introduces clever optimizations for ensuring that all markings in the tree are incomparable. At each step the new node is added to the tree only if its marking is not smaller than the marking of an existing node. Then, the tree is pruned: each node labelled with a marking that is smaller than the marking of the new node is removed together with all its successors. The idea is that a node that is not added or that is removed from the tree should be covered by the new node or one of its successors. It was recently shown that the MCT algorithm is incomplete [7]. The flaw is intricate and, according to [7], difficult to patch. As an illustration, an attempt to resolve this issue has been done in [9]. However, as proved in [6], the algorithm proposed in [9] may not terminate. In [7], an alternative algorithm, the CoverProc algorithm, is proposed for the computation of the MCS of a Petri net. This algorithm follows a different approach and is not based on the K&M Algorithm.

We propose here the Monotone-Pruning algorithm (MP), an improved K&M algorithm with pruning. This algorithm can be viewed as the MCT Algorithm with a slightly more aggressive pruning strategy which ensures completeness. The MP algorithm constitutes a simple modification of the K&M algorithm, and is thus easily amenable to implementation and to extensions to other classes of systems [1,3,4]. Moreover, as K&M Algorithm, and unlike the algorithm proposed in [7], any strategy of exploration of the Petri net is correct: depth first, breadth first, random... It is thus possible to develop heuristics for subclasses of Petri nets. Finally experimental results show that our algorithm is a strong improvement over the K&M Algorithm, it indeed dramatically reduces the exploration tree. In addition, MP Algorithm is also amenable to optimizations based on symbolic computations, as proposed in [5] for MCT.

While the algorithm in itself is simple and includes the elegant ideas of the original MCT Algorithm, the proof of its correctness is long and technical. In fact, the difficult part is its completeness, *i.e.* any reachable marking is covered by an element of the set returned by the algorithm. To overcome this difficulty, we reduce the problem to the correctness of the algorithm for a particular class of finite state systems, which we call widened Petri nets (WPN). These are Petri nets whose semantics is widened w.r.t. a given marking m: as soon as the number of tokens in a place p is greater than $m(p)$, this value is replaced by ω. Widened Petri nets generate finite state systems for which the proof of correctness of the Monotone-Pruning algorithm is easier as accelerations can be expressed as finite sequences of transitions.

Definitions of Petri nets and widened Petri nets are given in Section 2, together with the notions of minimal coverability set and reachability tree. In Section 3, we recall the K&M Algorithm and present the Monotone-Pruning Algorithm. We compare it with the MCT Algorithm, and prove its termination and correctness under the assumption that it is correct on WPN. Finally, in Section 4, we develop the proof of its correctness on widened Petri nets. Experimental results are given in Section 5. Omitted proofs can be found in [10].

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers. To denote that the union of two sets X and Y is disjoint, we write $X \uplus Y$. A quasi order $\leq$ on a set S is a reflexive and transitive relation on S. Given a quasi order $\leq$ on S, a state $s \in S$ and a subset X of S, we write $s \leq X$ iff there exists an element $s' \in X$ such that $s \leq s'$.

Given a finite alphabet Σ, we denote by Σ^* the set of words on Σ, and by ε the empty word. We denote by $\prec$ the (strict) prefix relation on Σ^*: given $u, v \in \Sigma^*$ we have $u \prec v$ iff there exists $w \in \Sigma^*$ such that $uw = v$ and $w \neq \varepsilon$. We denote by $\preceq$ the relation obtained as $\prec \cup =$.

2.1 Markings, ω-Markings and Labelled Trees

Given a finite set P, a *marking on* P is an element of the set $\mathsf{Mark}(P) = \mathbb{N}^P$. The set $\mathsf{Mark}(P)$ is naturally equipped with a partial order denoted $\leq$.

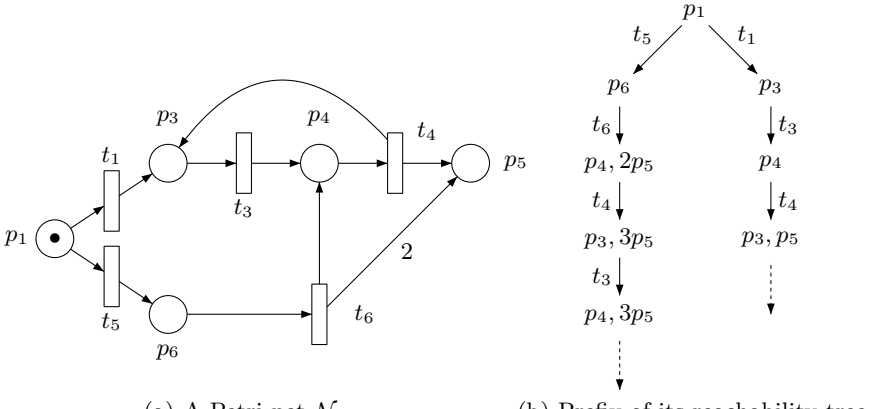
Given a marking $m \in \mathsf{Mark}(P)$, we represent it by giving only the positive components. For instance, $(1, 0, 0, 2)$ on $P = (p_1, p_2, p_3, p_4)$ is represented by the multiset $\{p_1, 2p_4\}$. An *ω-marking on* P is an element of the set $\mathsf{Mark}^\omega(P) = (\mathbb{N} \cup \{\omega\})^P$. The order $\leq$ on $\mathsf{Mark}(P)$ is naturally extended to this set by letting $n < \omega$ for any $n \in \mathbb{N}$, and $\omega \leq \omega$. Addition and substraction on $\mathsf{Mark}^\omega(P)$ is obtained using the rules $\omega + n = \omega - n = \omega$ for any $n \in \mathbb{N}$. The ω-marking $(\omega, 0, 0, 2)$ on $P = (p_1, p_2, p_3, p_4)$ is represented by the multiset $\{\omega p_1, 2p_4\}$.

Given two sets Σ_1 and Σ_2, a labelled tree is a tuple $\mathcal{T} = (N, n_0, E, \Lambda)$ where N is the set of nodes, $n_0 \in N$ is the root, $E \subseteq N \times \Sigma_2 \times N$ is the set of edges labelled with elements of Σ_2, and $\Lambda : N \to \Sigma_1$ labels nodes with elements of Σ_1. We extend the mapping Λ to sets of nodes: for $S \subseteq N$, $\Lambda(S) = \{\Lambda(n) \mid n \in S\}$. Given a node $n \in N$, we denote by $\mathsf{Ancestor}_{\mathcal{T}}(n)$ the set of ancestors of n in $\mathcal{T}$ (n included). If n is not the root of $\mathcal{T}$, we denote by $\mathsf{parent}_{\mathcal{T}}(n)$ its first ancestor in $\mathcal{T}$. Finally, given two nodes x and y such that $x \in \mathsf{Ancestor}_{\mathcal{T}}(y)$, we denote by $\mathsf{path}_{\mathcal{T}}(x, y) \in E^*$ the sequence of edges leading from x to y in $\mathcal{T}$. We also denote by $\mathsf{pathlabel}_{\mathcal{T}}(x, y) \in \Sigma_2^*$ the label of this path.

2.2 Petri Nets

Definition 1 (Petri nets (PN)). *A Petri net $\mathcal{N}$ is a tuple (P, T, I, O, m_0) where P is a finite set of* places*, T is a finite set of* transitions *with $P \cap T = \emptyset$, $I : T \to \mathsf{Mark}(P)$ is the backward incidence mapping, representing the* input *tokens, $O : T \to \mathsf{Mark}(P)$ is the forward incidence mapping, representing* output *tokens, and $m_0 \in \mathsf{Mark}(P)$ is the initial marking.*

The semantics of a PN is usually defined on markings, but can easily be extended to ω-markings. We define the semantics of $\mathcal{N} = (P, T, I, O, m_0)$ by its associated labeled transition system $(\mathsf{Mark}^\omega(P), m_0, \Rightarrow)$ where $\Rightarrow \subseteq \mathsf{Mark}^\omega(P) \times \mathsf{Mark}^\omega(P)$ is the transition relation defined by $m \Rightarrow m'$ iff $\exists t \in T$ s.t. $m \geq I(t) \wedge m' = m - I(t) + O(t)$. For convenience we will write, for $t \in T$, $m \overset{t}{\Longrightarrow} m'$ if $m \geq I(t)$ and $m' = m - I(t) + O(t)$. In addition, we also write $m' = \mathsf{Post}(m, t)$, this defines the operator Post which computes the successor of an ω-marking by a transition. We naturally extend this operator to sequences of transitions. Given an ω-marking m and a transition t, we write $m \overset{t}{\Rightarrow} \cdot$ iff there exists $m' \in \mathsf{Mark}^\omega(P)$ such that $m \overset{t}{\Rightarrow} m'$. The relation $\Rightarrow^*$ represents the reflexive and transitive closure of $\Rightarrow$. We say that a marking m is *reachable in* $\mathcal{N}$ iff $m_0 \Rightarrow^* m$. We say that a Petri net is bounded if the set of reachable markings is finite.



(a) A Petri net $\mathcal{N}$. (b) Prefix of its reachability tree.

Fig. 1. A Petri net with its reachability tree

Example 1. We consider the Petri net $\mathcal{N}$ depicted on Figure 1(a), which is a simplification of the counter-example proposed in [7], but is sufficient to present our definitions. The initial marking is $\{p_1\}$, depicted by the token in the place p_1. This net is not bounded as place p_5 can contain arbitrarily many tokens. The execution of the Monotone-Pruning algorithm on the original counter-example of [7] can be found in [10]. ⌟

2.3 Minimal Coverability Set of Petri Nets

We recall the definition of minimal coverability set introduced in [2].

Definition 2. *A coverability set of a Petri net $\mathcal{N} = (P, T, I, O, m_0)$ is a finite subset C of $\mathsf{Mark}^\omega(P)$ such that the two following conditions hold:*

1) *for every reachable marking m of $\mathcal{N}$, there exists $m' \in C$ such that $m \leq m'$,*
2) *for every $m' \in C$, either m' is reachable in $\mathcal{N}$ or there exists an infinite strictly increasing sequence of reachable markings $(m_n)_{n \in \mathbb{N}}$ converging to m'.*

A coverability set is minimal iff no proper subset is a coverability set. We denote by $\mathrm{MCS}(\mathcal{N})$ *the minimal coverability set of* $\mathcal{N}$.

Note that every two elements of a minimal coverability set are incomparable. Computing a minimal coverability set from a coverability set is easy. Note also that if the PN is bounded, then a set is a coverability set iff it contains all maximal reachable markings.

Example 2 (Example 1 continued). The MCS of the Petri net $\mathcal{N}$ is composed of the following ω-markings: $\{p_1\}$, $\{p_6\}$, $\{p_3, \omega p_5\}$, and $\{p_4, \omega p_5\}$.

2.4 Reachability Tree of Petri Nets

We recall the notion of reachability tree for a PN. This definition corresponds to the execution of the PN as a labelled tree. We require it to be coherent with the semantics of PN (soundness), to be complete w.r.t. the fireable transitions, and to contain no repetitions. Naturally, if the PN has an infinite execution, then this reachability tree is infinite.

Definition 3 (Reachability tree of a PN). *The reachability tree of a* PN $\mathcal{N} = (P, T, I, O, m_0)$ *is (up to isomorphism) a labelled tree* $\mathcal{R} = (N, n_0, E, \Lambda)$, *with* $E \subseteq N \times T \times N$ *and* $\Lambda : N \to \mathsf{Mark}(P)$, *s. t.:*

Root: $\Lambda(n_0) = m_0$,
Sound: $\forall (n, t, n') \in E, \Lambda(n) \overset{t}{\Rightarrow} \Lambda(n')$,
Complete: $\forall n{\in}N, \forall t{\in}T, \left(\exists m{\in}\mathsf{Mark}(P) \mid \Lambda(n) \overset{t}{\Rightarrow} m\right) \Rightarrow (\exists n'{\in}N \mid (n, t, n'){\in}E)$
Uniqueness: $\forall n, n', n'' \in N, \forall t \in T, (n, t, n') \in E \wedge (n, t, n'') \in E \Rightarrow n' = n''$

Using notations introduced for labelled trees, the following property holds:

Lemma 1. $\forall x, y \in N, x \in \mathsf{Ancestor}_{\mathcal{R}}(y) \Rightarrow \Lambda(y){=}\mathsf{Post}(\Lambda(x), \mathsf{pathlabel}_{\mathcal{R}}(x, y))$.

Example 3 (Example 1 continued). A prefix of the reachability tree of $\mathcal{N}$ is depicted on Figure 1(b). Each node is represented by its label (a marking). ⌟

2.5 Widened Petri Nets

We present an operation which, given a (potentially unbounded) Petri net, turns it into a finite state system. Let P be a finite set, and $\varphi \in \mathsf{Mark}(P)$ be a marking. We consider the *finite* set of ω-markings whose finite components (*i.e.* values different from ω) are less or equal than φ. Formally, we define $\mathsf{Mark}^{\omega}_{\varphi}(P) = \{m \in \mathsf{Mark}^{\omega}(P) \mid \forall p \in P, m(p) \leq \varphi(p) \vee m(p) = \omega\}$. The widening operator Widen_{φ} maps an ω-marking into an element of $\mathsf{Mark}^{\omega}_{\varphi}(P)$: $\forall m \in \mathsf{Mark}^{\omega}(P)$,

$$\forall p \in P, \mathsf{Widen}_{\varphi}(m)(p) = \begin{cases} m(p) & \text{if } m(p) \leq \varphi(p) \\ \omega & \text{otherwise.} \end{cases}$$

Note that this operator trivially satisfies $m \leq \mathsf{Widen}_{\varphi}(m)$.

Definition 4 (Widened Petri net). *A widened Petri net (*WPN *for short) is a pair* $(\mathcal{N}, \varphi)$ *composed of a* PN $\mathcal{N} = (P, T, I, O, m_0)$ *and of a marking* $\varphi \in \mathsf{Mark}(P)$ *such that* $m_0 \leq \varphi$.

The semantics of $(\mathcal{N}, \varphi)$ is given by its associated labelled transition system $(\mathsf{Mark}^{\omega}_{\varphi}(P), m_0, \Rightarrow_{\varphi})$ where for $m, m' \in \mathsf{Mark}^{\omega}_{\varphi}(P)$, and $t \in T$, we have $m \overset{t}{\Rightarrow}_{\varphi} m'$ iff $m' = \mathsf{Widen}_{\varphi}(\mathsf{Post}(m, t))$. We carry over from PN to WPN the notions of reachable marking, reachability tree... We define the operator Post_{φ} by $\mathsf{Post}_{\varphi}(m, t) = \mathsf{Widen}_{\varphi}(\mathsf{Post}(m, t))$. Subscript φ may be omitted when it is clear from the context. Finally, the minimal coverability set of a widened Petri net $(\mathcal{N}, \varphi)$ is simply the set of its maximal reachable states as its reachability set is finite. It is denoted $\mathrm{MCS}(\mathcal{N}, \varphi)$.

We state the following result, whose proof easily follows by induction.

Proposition 1. *Let* $(\mathcal{N}, \varphi)$ *be a* WPN*, and* m *be a reachable marking of* $\mathcal{N}$*. Then there exists an* ω*-marking* m' *reachable in* $(\mathcal{N}, \varphi)$ *such that* $m \leq m'$.

Example 4 (Example 1 continued). Consider the mapping φ associating 1 to places p_1, p_3, p_4 and p_6, and 2 to place p_5, and the widened Petri net $(\mathcal{N}, \varphi)$. Then for instance from marking $\{p_5, p_6\}$, the firing of t_6 results in the marking $\{p_4, \omega p_5\}$, instead of the marking $\{p_4, 3p_5\}$ in the standard semantics. Similarly, consider the prefix of the reachability tree of $\mathcal{N}$ depicted on Figure 1(b). For $(\mathcal{N}, \varphi)$, the reachability tree is obtained by substituting the marking $\{p_3, 3p_5\}$ (resp. $\{p_4, 3p_5\}$) with the ω-marking $\{p_3, \omega p_5\}$ (resp. $\{p_4, \omega p_5\}$), as we have $\varphi(p_5) = 2$. One can compute the MCS of this WPN. Due to the choice of φ, it coincides with the MCS of $\mathcal{N}$. ⌟

3 Monotone-Pruning Algorithm

3.1 Karp and Miller Algorithm

The K&M Algorithm [8] is a well known solution to compute a coverability set of a PN. It is represented as Algorithm 1 (with a slight modification as in [8], the algorithm computes simultaneously all the successors of a marking). K&M algorithm uses an external acceleration function $\mathsf{Acc} : 2^{\mathsf{Mark}^{\omega}(P)} \times \mathsf{Mark}^{\omega}(P) \to \mathsf{Mark}^{\omega}(P)$ which is defined as follows:

$$\forall p \in P, \mathsf{Acc}(M, m)(p) = \begin{cases} \omega & \text{if } \exists m' \in M \mid m' < m \wedge m'(p) < m(p) < \omega \\ m(p) & \text{otherwise.} \end{cases}$$

K&M Algorithm builds a tree in which nodes are labelled by ω-markings and edges by transitions of the Petri net. Roughly, it consists in exploring the reachability tree of the PN, and in applying the acceleration function Acc on branches of this tree. Note that the acceleration may compute ω-markings that are not reachable. The correctness of this procedure relies on the strict monotonicity of PN and on the fact that the order $\leq$ on ω-markings is well-founded.

Theorem 1 ([8]). *Let $\mathcal{N}$ be a* PN. K&M *algorithm terminates and computes a coverability set of $\mathcal{N}$.*

Algorithm 1. The K&M Algorithm

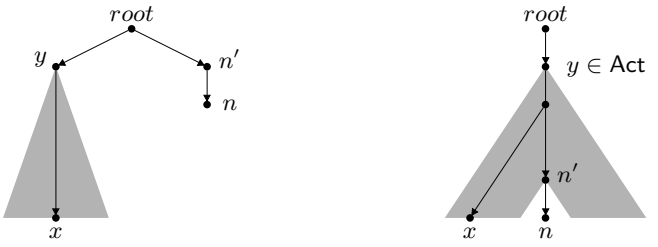
Require: A Petri net $\mathcal{N} = (P, T, I, O, m_0)$.
Ensure: A labelled tree $\mathcal{C} = (X, x_0, B, \Lambda)$ such that X is a coverability set of $\mathcal{N}$.
1: Let x_0 be a new node such that $\Lambda(x_0) = m_0$.
2: $X := \{x_0\}$; $\mathsf{Wait} := \{(x_0, t) \mid \Lambda(x_0) \stackrel{t}{\Rightarrow} \cdot\}$; $B := \emptyset$;
3: **while** $\mathsf{Wait} \neq \emptyset$ **do**
4: Pop (n', t) from Wait. $m := \mathsf{Post}(\Lambda(n'), t)$;
5: **if** $\nexists y \in \mathsf{Ancestor}_{\mathcal{C}}(n') \mid \Lambda(y) = m$ **then**
6: Let n be a new node s.t. $\Lambda(n) = \mathsf{Acc}(\Lambda(\mathsf{Ancestor}_{\mathcal{C}}(n')), m)$;
7: $X = X \cup \{n\}$; $B = B \cup \{(n', t, n)\}$; $\mathsf{Wait} = \mathsf{Wait} \cup \{(n, u) \mid \Lambda(n) \stackrel{u}{\Rightarrow} \cdot\}$;
8: **end if**
9: **end while**
10: Return $\mathcal{C} = (X, x_0, B, \Lambda)$.

3.2 Definition of the Algorithm

We present in this section our algorithm which we call Monotone-Pruning Algorithm as it includes a kind of horizontal pruning. We denote this algorithm by MP. It involves the acceleration function Acc used in the Karp and Miller algorithm. However, it is applied in a slightly different manner.

Algorithm 2. Monotone Pruning Algorithm for Petri Nets.

Require: A Petri net $\mathcal{N} = (P, T, I, O, m_0)$.
Ensure: A labelled tree $\mathcal{C} = (X, x_0, B, \Lambda)$ and a partition $X = \mathsf{Act} \biguplus \mathsf{Inact}$ such that $\Lambda(\mathsf{Act}) = \mathrm{MCS}(\mathcal{N})$.
1: Let x_0 be a new node such that $\Lambda(x_0) = m_0$;
2: $X := \{x_0\}$; $\mathsf{Act} := X$; $\mathsf{Wait} := \{(x_0, t) \mid \Lambda(x_0) \stackrel{t}{\Rightarrow} \cdot\}$; $B := \emptyset$;
3: **while** $\mathsf{Wait} \neq \emptyset$ **do**
4: Pop (n', t) from Wait.
5: **if** $n' \in \mathsf{Act}$ **then**
6: $m := \mathsf{Post}(\Lambda(n'), t)$;
7: Let n be a new node such that $\Lambda(n) = \mathsf{Acc}(\Lambda(\mathsf{Ancestor}_{\mathcal{C}}(n') \cap \mathsf{Act}), m)$;
8: $X = X \cup \{n\}$; $B = B \cup \{(n', t, n)\}$;
9: **if** $\Lambda(n) \not\leq \Lambda(\mathsf{Act})$ **then**
10: $\mathsf{Act} = \mathsf{Act} \setminus \{x \mid \exists y \in \mathsf{Ancestor}_{\mathcal{C}}(x). \Lambda(y) \leq \Lambda(n) \wedge (y \in \mathsf{Act} \vee y \notin \mathsf{Ancestor}_{\mathcal{C}}(n))\}$;
11: $\mathsf{Act} = \mathsf{Act} \cup \{n\}$; $\mathsf{Wait} = \mathsf{Wait} \cup \{(n, u) \mid \Lambda(n) \stackrel{u}{\Rightarrow} \cdot\}$;
12: **end if**
13: **end if**
14: **end while**
15: Return $\mathcal{C} = (X, x_0, B, \Lambda)$ and $(\mathsf{Act}, \mathsf{Inact})$.



(a) $y \notin \mathsf{Ancestor}_{\mathcal{C}}(n)$ (b) $y \in \mathsf{Ancestor}_{\mathcal{C}}(n) \cap \mathsf{Act}$

Fig. 2. Deactivations of MP Algorithm

As Karp and Miller Algorithm, MP Algorithm builds a tree $\mathcal{C}$ in which nodes are labelled by ω-markings and edges by transitions of the Petri net. Therefore it proceeds in an exploration of the reachability tree of the Petri net, and uses acceleration along branches to reach the "limit" markings. In addition, it can prune branches that are covered by nodes on other branches. Therefore, nodes of the tree are partitionned in two subsets: active nodes, and inactive ones. Intuitively, active nodes will form the minimal coverability set of the Petri net, while inactive ones are kept to ensure completeness of the algorithm.

Given a pair (n', t) popped from Wait, the introduction in $\mathcal{C}$ of the new node obtained from (n', t) proceeds in the following steps:

1. node n' should be active (test of Line 5) ;
2. the "regular" successor marking is computed: $m = \mathsf{Post}(\Lambda(n'), t)$ (Line 6) ;
3. this marking is accelerated w.r.t. the *active ancestors* of node n', and a new node n is created with this marking: $\Lambda(n) = \mathsf{Acc}(\Lambda(\mathsf{Ancestor}_{\mathcal{C}}(n') \cap \mathsf{Act}), m)$ (Lines 7 and 8) ;
4. the new node n is declared as active if, and only if, it is not covered by an existing active node (test of Line 9 and Line 11) ;
5. update of Act: some nodes are "deactivated" (Line 10).

We detail the update of the set Act. Intuitively, one wants to deactivate nodes (and their descendants) that are covered by the new node n. This would lead to deactivate a node x iff it owns an ancestor y dominated by n, *i.e.* such that $\Lambda(y) \leq \Lambda(n)$. This condition has to be refined to obtain a correct algorithm (see Remark 1). In MP Algorithm (see Line 10), node x is deactivated iff its ancestor y is either active ($y \in \mathsf{Act}$), or is not itself an ancestor of n ($y \notin \mathsf{Ancestor}_{\mathcal{C}}(n)$). In this case, we say that x *is deactivated by* n. This subtle condition constitutes the main difference between MP and MCT Algorithms (see Remark 1).

Consider the introduction of a new node n obtained from $(n', t) \in \mathsf{Wait}$, and a node y such that $\Lambda(y) \leq \Lambda(n)$, y can be used to deactivate nodes in two ways:

- if $y \notin \mathsf{Ancestor}_{\mathcal{C}}(n)$, then no matter whether y is active or not, all its descendants are deactivated (represented in gray on Figure 2(a)),
- if $y \in \mathsf{Ancestor}_{\mathcal{C}}(n)$, then y must be active ($y \in \mathsf{Act}$), and in that case all its descendants are deactivated, except node n itself as it is added to Act at Line 11 (see Figure 2(b)).

The main result of the paper is that MP Algorithm terminates and is correct:

Theorem 2. *Let $\mathcal{N}$ be a* PN. MP *algorithm terminates and computes a minimal coverability set of $\mathcal{N}$.*

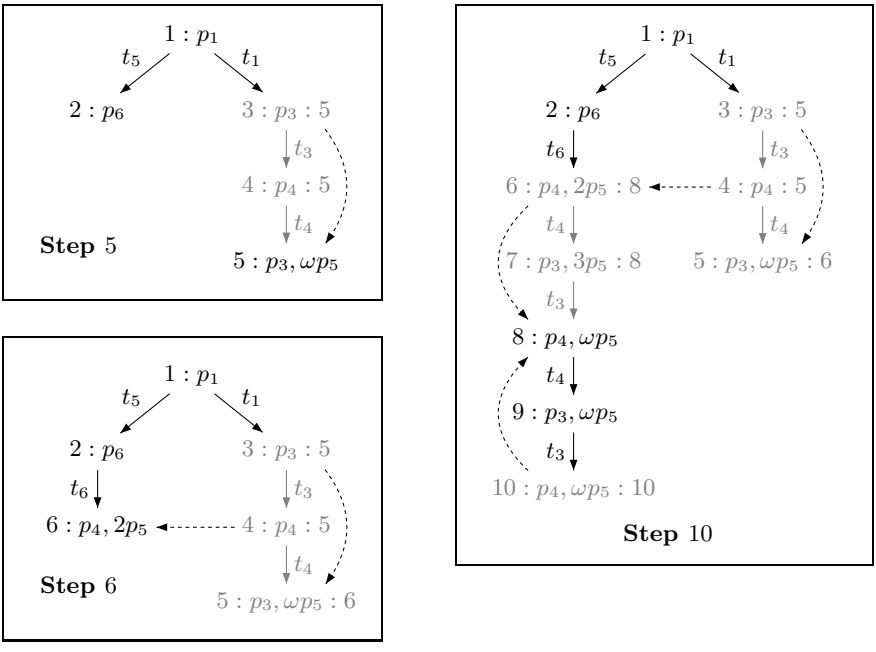


Fig. 3. Snapshots of the execution of MP Algorithm on PN $\mathcal{N}$

Example 5 (Example 1 continued). We consider the execution of MP Algorithm on the PN $\mathcal{N}$. Three intermediary steps (5, 6 and 10) are represented on Figure 3. The numbers written on the left (before the separator ":") of nodes indicate the order in which nodes are created. Nodes that are deactivated are represented in light gray, and dashed arrows indicate how nodes are deactivated. In addition, the number of the node in charge of the deactivation is represented at the right (after the separator ":"). In the following explanations, node n_i denotes the node that has been created at step i:

- At step 5, the new node n_5 ($\{p_3, p_5\}$) covers node n_3 ($\{p_3\}$), which is thus deactivated, together with its descendants, except node n_5 that is just added.
- At step 6, the new node n_6 ($\{p_4, 2p_5\}$) covers node n_4 ($\{p_4\}$). This node was already deactivated but as it lies on another branch, it can be used to discard its descendants. As a consequence node n_5 is deactivated.
- At step 10, the new node n_{10} is covered by node n_8, which is still active. Thus n_{10} is immediately declared as inactive. ⌟

After step 10, MP terminates and the active nodes give the MCS of $\mathcal{N}$.

Remark 1 (Comparison with the MCT *Algorithm.).* One can verify that, except the fact that the MCT algorithm develops all successors of a node simultaneously (as K&M does), the MCT Algorithm can be obtained from the MP Algorithm by a subtle modification. The only difference comes from the deactivation of nodes. In MP, inactivate nodes can be used to deactivate nodes. In MCT, only active nodes are used to deactivate nodes. More precisely, MCT Algorithm is obtained by replacing Line 10 by the following line :

$10'$: $\quad \mathsf{Act}- = \{x \mid \exists y \in \mathsf{Ancestor}_{\mathcal{C}}(x).\Lambda(y) \leq \Lambda(n) \wedge y \in \mathsf{Act}\}$;

Thus, condition $(y \in \mathsf{Act} \vee y \notin \mathsf{Ancestor}_{\mathcal{C}}(n))$ in MP is replaced by the stronger condition $y \in \mathsf{Act}$ to obtain MCT. In particular, this shows that more nodes are pruned in MP Algorithm.

Note also that if one considers the trivial condition `true`, *i.e.* one considers all active and inactive nodes to discard nodes, then one looses the termination of the algorithm. Consider Example 5. With this condition, node n_9 covers node n_7 and thus deactivates node n_8 (this does not happen in MP as n_7 is an inactive ancestor of n_9). But then, node n_{10} covers n_8 and deactivates n_{10}, and so on.

*Remark 2 (*MP *Algorithm for widened Petri nets.).* In the sequel, we will consider the application of MP Algorithm on widened Petri nets. Let $(\mathcal{N}, \varphi)$ be a WPN. The only difference is that the operator Post (resp. $\Rightarrow$) must be replaced by the operator Post_φ (resp. $\Rightarrow_\varphi$). For WPN, MP Algorithm satisfies an additional property. One can prove by induction that all markings computed by MP are reachable in $(\mathcal{N}, \varphi)$. Indeed, the acceleration is consistent with the semantics of $(\mathcal{N}, \varphi)$, *i.e.* all markings computed by Acc belong to $\mathsf{Mark}^{\omega}_{\varphi}(P)$ (where P denotes the set of places of $\mathcal{N}$), provided the arguments of Acc do.

Example 6 (Example 4 continued). Consider the WPN $(\mathcal{N}, \varphi)$ introduced in Example 4. Running MP on this WPN also yields the trees depicted on Figure 3.

3.3 Termination of MP Algorithm

Theorem 3. MP *Algorithm terminates.*

Proof. We proceed by contradiction, and assume that the algorithm does not terminate. Let $\mathcal{C} = (X, x_0, B, \Lambda)$ and $X = \mathsf{Act} \uplus \mathsf{Inact}$ be the labelled tree and the partitions computed by MP. As $\mathcal{C}$ is of finite branching (bounded by $|T|$), there exists by König's lemma an infinite branch in this tree. We fix such an infinite branch, and write it $b = x_0 \xrightarrow{t_0} x_1 \xrightarrow{t_1} x_2 \ldots$, with $(x_i, t_i, x_{i+1}) \in B, \forall i$.

Let $n \in X \setminus \{x_i \mid i \geq 0\}$. We claim that n cannot deactivate any of the x_i's. By contradiction, if for some i, x_i is deactivated by n, then all the descendants of x_i are also deactivated, except n if it is a descendant of x_i. As n does not belong to b, this implies that for any $j \geq i$, x_j is deactivated. This is impossible because branch b is infinite and the algorithm only computes successors of active nodes (test of Line 5).

By definition of the acceleration function, two cases may occur: either one of the active ancestors is strictly dominated, and then a new ω will appear in the resulting marking $(\exists p \mid \Lambda(n)(p) = \omega > m(p))$, or no active ancestors is strictly

dominated, and then the acceleration has no effect on marking m ($\Lambda(n) = m$). We say that in the first case, there is an "effective acceleration".

By definition of the semantics of a Petri net on ω-markings, once a marking has value ω on a place p, so will all its successors. Thus, as there are finitely many places, a finite number of effective accelerations can occur on branch b. We consider now the largest suffix of the branch b containing no effective accelerations: let i be the smallest positive integer such that for any $j \geq i$, we have $\Lambda(x_{j+1}) = \mathsf{Post}(\Lambda(x_j), t_j)$.

We will prove that the set $S = \{x_j \mid j \geq i\}$ is an infinite set of active nodes with pairwise incomparable markings, which is impossible as the set $\mathsf{Mark}^\omega(P)$ equipped with partial order $\leq$ is a well-founded quasi-order, yielding the contradiction.

We proceed by induction and prove that for any $j \geq i$ the set $S_j = \{x_k \mid j \geq k \geq i\}$ contains active nodes with pairwise incomparable markings. Recall that we have shown above that nodes not on b cannot deactivate nodes of b. Consider set S_i. When node x_i is created, it must be declared as active, otherwise none of its successors are built, that is impossible as b is infinite. Let $j \geq i$, assume that property holds for S_j, and consider the introduction of node x_{j+1}. We prove that x_{j+1} is active, that it deactivates no node of S_j, and that the markings of S_{j+1} are pairwise incomparable. First, as for node x_i, the new node x_{j+1} must be declared as active when it is created. Thus no active node covers it (Line 9). By induction, elements of S_j are active, and thus we have $\Lambda(x_{j+1}) \not\leq \Lambda(x)$ for any $x \in S_j$. Second, as we have $\Lambda(x_{j+1}) = \mathsf{Post}(\Lambda(x_j), t_j)$, we do not use an effective acceleration, and thus x_{j+1} strictly dominates none of its active ancestors. In particular, this implies that it does not deactivate any of its ancestors, and thus any element of S_j. Moreover, by induction, elements of S_j are active nodes, thus $\Lambda(x_{j+1}) \not> \Lambda(x)$ for any $x \in S_j$: elements of S_{j+1} are pairwise incomparable. □

3.4 Correctness of MP Algorithm

We reduce the correctness of MP Algorithm for Petri nets to the correctness of this algorithm for widened Petri nets, which are finite state systems. This latter result is technical, and proved in the next section (see Theorem 5).

We use this theorem to prove:

Theorem 4. MP *Algorithm for Petri nets is correct.*

Proof. Let $\mathcal{N} = (P, T, I, O, m_0)$ be a PN, $\mathcal{C} = (X, x_0, B, \Lambda)$ be the labelled tree and $X = \mathsf{Act} \biguplus \mathsf{Inact}$ be the partition built by MP Algorithm on $\mathcal{N}$. As MP Algorithm terminates, all these objects are finite. We will prove that $\Lambda(\mathsf{Act})$ is the minimal coverability set of $\mathcal{N}$.

First note that elements of $\Lambda(\mathsf{Act})$ are pairwise incomparable: this is a simple consequence of Lines 9, 10 and 11. Thus, we only have to prove that it is a coverability set.

The soundness of the construction, *i.e.* the fact that elements of $\Lambda(\mathsf{Act})$ satisfy point 2 of Definition 2, follows from the correctness of the acceleration function. To prove the completeness, *i.e.* point 1 of Definition 2, we use the correctness of

MP Algorithm on widened Petri nets. We can consider, for each place $p \in P$, the largest value appearing in a marking during the computation. This defines a marking $\varphi \in \mathsf{Mark}(P)$.

We consider now the widened Petri net $(\mathcal{N}, \varphi)$ and the execution of MP Algorithm on it (see Remark 2). We claim that there exists an execution of this algorithm which builds the same labelled tree $\mathcal{C}$ and the same partition. This execution is obtained by picking the same elements in the list Wait. This property can be proven by induction on the length of the execution of the algorithm. Indeed, by definition of marking φ, operators Post and Post_φ are equivalent on the markings computed by the algorithm. Thus, both algorithms perform exactly the same accelerations and compute the same ω-markings.

By correctness of MP Algorithm on WPN (see Theorem 5), we obtain $\Lambda(\mathsf{Act}) = \mathrm{MCS}(\mathcal{N}, \varphi)$. By Proposition 1, any marking reachable in $\mathcal{N}$ is covered by a reachable marking of $(\mathcal{N}, \varphi)$, and thus by $\mathrm{MCS}(\mathcal{N}, \varphi) = \Lambda(\mathsf{Act})$. □

4 MP Algorithm for WPN

We devote this section to the proof that MP Algorithm is correct on WPN.

4.1 Outline

As for Petri nets, the main difficulty is to prove the completeness of the set returned by MP. Therefore, we introduce in Subsection 4.2 a notion of *exploration of a* WPN which corresponds to a tree built on the reachability tree of the WPN, with some additional properties. This structure allows to explicit the effect of accelerations. We prove in Subsection 4.3 that MP indeed builds an exploration. In fact, other algorithms like K&M or MCT also do build explorations. Finally, we prove that the exploration built by MP is complete in Subsection 4.4: we show that any reachable marking is covered by an active node. Therefore, we introduce a notion of covering path which intuitively explicits the sequence of transitions that remain to be fired from a given active node to reach the desired marking. We prove that such covering paths are not cyclic, that is promises are always fulfilled.

4.2 Exploration of a WPN

To build a coverability set the different algorithms we consider (K&M, MCT and MP) proceed in a similar way. Roughly, the algorithm starts with the root of the reachability tree and picks a fireable transition t. Then it picks a descendant that may either be the direct child by t (no acceleration) or a descendant obtained after skipping a few nodes (acceleration), this descendant must be strictly greater than the direct child (by t). Then if this node is not covered by the previously selected (and active) nodes, a pruning may occur (not in the K&M algorithm): some active nodes are deactivated, intuitively because the subtree rooted at the new node should cover those nodes. The process continues with active nodes.

This process can be viewed as an exploration of the reachability tree $\mathcal{R} = (N, n_0, E, \Lambda)$ in the following sense. We define below an exploration as

a tuple $\mathcal{E} = (X, B, \alpha, \beta)$, where X is the subset of N explored by the algorithm, B is an edge relation on X, such that $(x, t, x') \in B$ if x' is the node built by the algorithm when processing the transition t fireable from x. The function α gives the order in which nodes of X are explored by the algorithm. The function β gives the position at which a node is deactivated, *i.e.* $\beta(n) = i$ if n is deactivated (pruned) when the i-th node appears.

Definition 5 (Exploration). *Given a* WPN $(\mathcal{N}, \varphi)$ *and its reachability tree* $\mathcal{R} = (N, n_0, E, \Lambda)$*, an exploration of* $\mathcal{R}$ *is a tuple* $\mathcal{E} = (X, B, \alpha, \beta)$ *such that*

- X *is a finite subset of* N*,*
- $B \subseteq X \times T \times X$*,*
- $n_0 \in X$*,*
- $(X, n_0, B, \Lambda_{|X})$ *is a labelled tree,*
- α *is a bijection from* X *to* $\{1, \ldots, |X|\}$*, and*
- β *is a mapping from* X *to* $\{1, \ldots, |X|\} \cup \{+\infty\}$*.*

For any $1 \leq i \leq |X|$*, we define the sets* $X_i = \{x \in X \mid \alpha(x) \leq i\}$, $\mathsf{Inact}_i = \{x \in X \mid \beta(x) \leq i\}$*, and* $\mathsf{Act}_i = X_i \setminus \mathsf{Inact}_i$*. We let* $\mathsf{Act} = \mathsf{Act}_{|X|}$ *and* $\mathsf{Inact} = \mathsf{Inact}_{|X|}$*.*

In addition, we require the following conditions:

(i) $\alpha \leq \beta$,
(ii) $\forall x, y \in X,\ x \in \mathsf{Ancestor}_{\mathcal{R}}(y) \Rightarrow \alpha(x) \leq \alpha(y)$,
(iii) $\forall (x, t, y) \in B, \alpha(y) \leq \beta(x)$,
(iv) T**-completeness:** $\forall x \in \mathsf{Act}, \forall t \in T$ *s.t.* $\Lambda(x) \stackrel{t}{\Rightarrow}_{\varphi} \cdot, \exists y \in X \mid (x, t, y) \in B$,
(v) $\forall (x, t, y) \in B$*, there exists* $z \in N$ *such that:*
  (a) $(x, t, z) \in E$*, and* $z \in \mathsf{Ancestor}_{\mathcal{R}}(y)$,
  (b) $\mathsf{Post}_{\varphi}(\Lambda(x), t) = \Lambda(z) \leq \Lambda(y)$.

The first condition states that nodes cannot be deactivated strictly before being selected. The second condition states that nodes are selected downward: one cannot select a node that has a descendant already selected. Condition (iii) states that the algorithm explores subtrees of active nodes only. Condition (iv) enforces that all fireable transitions of active nodes are explored. The last condition (see Figure 4, where the cone below node z denotes the descendants of z in $\mathcal{R}$) requires that the selected descendant is either the direct child by the selected transition t or a descendant of this child whose marking is greater than the marking of the child (acceleration). In the sequel, we denote by $\mathsf{Ancestor}_{\mathcal{E}}(\cdot)$ the ancestor relation considered in the labelled tree $(X, n_0, B, \Lambda_{|X})$. By definition, we have the following simple property: $\forall x \in X, \mathsf{Ancestor}_{\mathcal{E}}(x) = \mathsf{Ancestor}_{\mathcal{R}}(x) \cap X$.

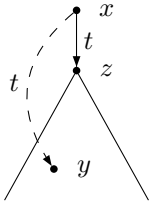


Fig. 4. Condition $(v).(a)$ of Def. 5

It is easy to verify that sets Act_i and Inact_i form a partition of X_i ($X_i = \mathsf{Act}_i \biguplus \mathsf{Inact}_i$) and that sets Inact_i are increasing ($\mathsf{Inact}_i \subseteq \mathsf{Inact}_{i+1}, \forall i < |X|$).

Remark 3. A trivial case of exploration is obtained when relation B coincides with the restriction of relation E to the set X. This case in fact corresponds to the exploration obtained by an algorithm that would perform no acceleration.

Remark 4. It can be proven that K&M and MCT applied on WPN do build explorations. Consider K&M Algorithm. As it deactivates no node, it yields $\beta(n) = +\infty$ for any node n. However, it uses some accelerations and therefore some nodes are skipped but it respects condition (v).

4.3 MP-Exploration of a WPN

Let $(\mathcal{N}, \varphi)$ be a WPN with $\mathcal{N} = (P, T, I, O, m_0)$, and $\mathcal{C} = (X, x_0, B, \Lambda)$, $X = \mathsf{Act} \uplus \mathsf{Inact}$ be the labelled tree and the partition returned by the MP Algorithm. We define here the two mappings α and β that allow to show that the labelled tree $\mathcal{C}$ can be interpreted as an exploration in the sense of Definition 5.

Mapping α. It is simply defined as the order in which elements of X are built by the MP Algorithm.

Mapping β. Initially, the set Act contains the node x_0. Any new node n can be added only once in set Act, immediately when it is added in X (Line 11) (and can thus be removed from Act at most once). We define mapping β as follows:

- if a node x never enters set Act, then we let $\beta(x) = \alpha(x)$.
- if a node x enters set Act and is never removed, then we let $\beta(x) = +\infty$.
- finally, in the last case, let x be a node which enters set Act and is removed from it during the execution of the algorithm. Nodes can only be removed from set Act at Line 10. Then let n be the node added to X at Line 8 during this execution of the **while** loop, we define $\beta(x) = \alpha(n)$.

Remark 5. Using these definitions of mappings α and β, one can verify that intermediary values of sets X and Act computed by the algorithm coincide with sets X_i and Act_i defined in Definition 5.

Example 7 (Example 6 continued). On Figure 3, numbers indicated on the left and on the right of nodes correspond to values of mappings α and β. When no number is written on the right, this means that the node is active, and then the value of β is $+\infty$.

Embedding of $\mathcal{C} = (X, x_0, B, \Lambda)$ in the reachability tree. In the labelled tree $\mathcal{C}$ built by the algorithm, the label of the new node n obtained from the pair (n', t) is computed by function Acc. To prove that $\mathcal{C}$ can be embedded in the reachability tree of $(\mathcal{N}, \varphi)$, we define a mapping called the concretization function which expresses the marking resulting from the acceleration as a marking reachable in $(\mathcal{N}, \varphi)$ from marking $\Lambda(n')$. Intuitively, an acceleration represents the repetition of some sequences of transitions until the upper bound is reached. As the system is finite (we consider widened Petri nets), we can exhibit a particular sequence of transitions which allows to reach this upper bound.

Definition 6 (Concretization function). *The concretization function is a mapping γ from B^* to T^*. Given a sequence of adjacent edges $b_1 \dots b_k \in B$, we define $\gamma(b_1 \dots b_k) = \gamma(b_1) \dots \gamma(b_k)$. We let $M = \max\{\varphi(p) \mid p \in P\} + 1$.*

Let $b = (n', t, n) \in B$. The definition of γ proceeds by induction on $\alpha(n')$: we assume γ is defined on all edges $(x, u, y) \in B$ such that $\alpha(x) < \alpha(n')$.

Let $m = \mathsf{Post}_\varphi(\Lambda(n'), t)$, then there are two cases, either :

1. *$\Lambda(n) = m$ (t is not accelerated), then we define $\gamma(b) = t$, or*
2. *$\Lambda(n) > m$. Let $X' = \{x_1, \dots, x_k\}$ (x_i's are ordered w.r.t. α) defined by: $X' = \{x \in \mathsf{Ancestor}_\mathcal{C}(n') \cap \mathsf{Act}_{\alpha(n)-1} \mid \Lambda(x) \leq m \wedge \exists p. \Lambda(x)(p) < m(p) < \omega\}$. For each $j \in \{1, \dots, k\}$, let $w_j = \mathsf{path}_\mathcal{C}(x_j, n') \in B^*$. Then we define: $\gamma(b) = t.(\gamma(w_1).t)^M \dots (\gamma(w_k).t)^M$.*

We prove in [10] the following Lemma which states the expected property of the concretization function:

Lemma 2. *Let $x, y \in X$ such that $x \in \mathsf{Ancestor}_\mathcal{C}(y)$, and let $w = \mathsf{path}_\mathcal{C}(x, y)$. Then we have $\mathsf{Post}_\varphi(\Lambda(x), \gamma(w)) = \Lambda(y)$.*

This result allows to prove by induction that the labelled tree built by MP is, up to an isomorphism, included in the reachability tree of the WPN (see details in [10]), and is thus an exploration:

Proposition 2 (MP-exploration). *The execution of* MP *Algorithm on a* WPN *$(\mathcal{N}, \varphi)$ defines an exploration $\mathcal{E}$ of $(\mathcal{N}, \varphi)$. We call this exploration an* MP-exploration of $(\mathcal{N}, \varphi)$.

4.4 Main Proof

Theorem 5. MP *Algorithm for* WPN *terminates and computes the* MCS.

Termination of MP for WPN can be proved as in Theorem 3. As a consequence of Lemma 2, MP algorithm only computes markings that are reachable in the WPN, therefore the algorithm is sound. We devote the rest of this section to the proof of its completeness.

Fix a WPN $(\mathcal{N}, \varphi)$, with $\mathcal{N} = (P, T, I, O, m_0)$, and let $\mathcal{E} = (X, B, \alpha, \beta)$ be an MP-exploration of $(\mathcal{N}, \varphi)$. We will use notations X, Act and Inact of Definition 5.

Preliminary properties. Given a node $n \in X$, we define the predicate $\mathsf{disc}(n)$ as $\beta(n) = \alpha(n)$. When this holds, we say that n is discarded as it is immediately added to the set Inact. In that case, no other node is deactivated.

Given two nodes $n, x \in X$ such that $\alpha(n) \leq \alpha(x)$ and $n \in \mathsf{Inact}$, we define the predicate $\mathsf{prune}(n, x)$ as $\exists y \in \mathsf{Ancestor}_\mathcal{E}(n). \Lambda(y) \leq \Lambda(x) \wedge (y \in \mathsf{Act}_{\beta(n)-1} \vee y \notin \mathsf{Ancestor}_\mathcal{E}(x))$.

One can check that the MP-exploration $\mathcal{E}$ satisfies the following properties. Arbitrary explorations do not satisfy them.

Proposition 3. *Let $n \in \mathsf{Inact}$, then:*

(i) $\mathsf{disc}(n) \iff \Lambda(n) \leq \mathsf{Act}_{\alpha(n)-1}$.
(ii) $\neg\mathsf{disc}(n) \Rightarrow \mathsf{prune}(n, x)$, *where* $x = \alpha^{-1}(\beta(n))$.
(iii) $\forall x \in X$ *s.t.* $\alpha(n) \leq \alpha(x)$, *if* $\mathsf{prune}(n, x) \wedge \neg\mathsf{disc}(x)$, *then* $\beta(n) \leq \alpha(x)$

Covering Function. We introduce a function $\mathsf{Temp\text{-}Cover}$ which explicits why nodes are deactivated. Intuitively, for a node $n \in \mathsf{Inact}$, if we have $\mathsf{Temp\text{-}Cover}(n) = (x, \varrho) \in X \times T^*$, this means that node x is in charge of deactivation of n, and that the firing of the sequence ϱ from $\Lambda(x)$ leads to a state dominating $\Lambda(n)$. Note that to identify the sequence in T^*, we use the path between nodes *in the reachability tree*. This is possible as by definition, the exploration is embedded in the reachability tree.

Definition 7 ($\mathsf{Temp\text{-}Cover}$). *The mapping* $\mathsf{Temp\text{-}Cover}$ *is defined from* Inact *to* $X \times T^*$ *as follows. Let* $n \in \mathsf{Inact}$*, and* $i = \beta(n)$*. We distinguish two cases:*

Discarded: *If* $\mathsf{disc}(n)$*, then by Proposition 3.*(i)*, there exists a node* $x \in \mathsf{Act}_{i-1}$ *such that* $\Lambda(n) \leq \Lambda(x)$*, we define*[1] $\mathsf{Temp\text{-}Cover}(n) = (x, \varepsilon)$.

Not discarded: *Otherwise,* $\neg\mathsf{disc}(n)$ *holds. By Proposition 3.*(ii)*,* $\mathsf{prune}(n, x)$ *holds, where* $x = \alpha^{-1}(i)$*. We fix*[2] *a witness* y *of property* $\mathsf{prune}(n, x)$*, and let* $\varrho = \mathsf{pathlabel}_{\mathcal{R}}(y, n) \in T^*$*. We define* $\mathsf{Temp\text{-}Cover}(n) = (x, \varrho)$.

The following property easily follows from Definition 7 and Lemma 1:

Lemma 3. *Let* $n \in \mathsf{Inact}$*,* $\mathsf{Temp\text{-}Cover}(n) = (x, \varrho)$*. Then* $\Lambda(n) \leq \mathsf{Post}_{\varphi}(\Lambda(x), \varrho)$.

The previous definition is temporary, in the sense that it describes how a node is deactivated. However, active nodes may be deactivated, and thus nodes referenced by mapping $\mathsf{Temp\text{-}Cover}$ may not belong to set Act. In order to recover an active node from which a dominating node can be obtained, we define a mapping which records for each inactivate node the successive covering informations:

Definition 8 (Covering function). *The covering function* Cover *is a mapping from* X *to* $(X \times T^*)^*$*. It is recursively defined as follows. Let* $n \in X$.

1. *if* $n \in \mathsf{Act}$*, then* $\mathsf{Cover}(n) = \varepsilon$;
2. *otherwise, let* $\mathsf{Temp\text{-}Cover}(n) = (x, \varrho)$*. We define* $\mathsf{Cover}(n) = (x, \varrho) \cdot \mathsf{Cover}(x)$.

Example 8 (Example 6 continued). We illustrate the definition of the covering function on Example 6. MP Algorithm terminates at step 10. Consider node n_3, deactivated at step 5. We have $\mathsf{Temp\text{-}Cover}(n_3) = (n_5, \varepsilon)$. Indeed, it is directly covered by node n_5. Node n_5 is deactivated at step 6 by node n_6 through node n_4, which is its ancestor by transition t_4, then we have $\mathsf{Temp\text{-}Cover}(n_5) = (n_6, t_4)$. Node n_6 is deactivated at step 8 because it is directly covered by node n_8, thus we have $\mathsf{Temp\text{-}Cover}(n_6) = (n_8, \varepsilon)$. We finally obtain $\mathsf{Cover}(n_3) = (n_5, \varepsilon) \cdot (n_6, t_4) \cdot (n_8, \varepsilon)$. One can verify that $\Lambda(n_3) \leq \mathsf{Post}_{\varphi}(\Lambda(n_8), t_4)$. □

We state the next property which follows from Lemma 3 by induction:

Lemma 4. *Let* $n \in \mathsf{Inact}$ *be such that* $\mathsf{Cover}(n) = (x_1, \varrho_1) \cdots (x_k, \varrho_k)$*. Then* $\Lambda(n) \leq \mathsf{Post}_{\varphi}(\Lambda(x_k), \varrho_k \varrho_{k-1} \ldots \varrho_1)$.

[1] We choose any such node x.
[2] We could pick any such node y.

We now state a core property of mapping Cover, holding for MP-explorations. It is fundamental to prove the absence of cycles, and thus the fact that the exploration yields a minimal coverability set. Roughly, it states that intermediary markings skipped by accelerations would not modify activations/deactivations:

Proposition 4. *Let $x \in \mathsf{Inact}$ be such that $\mathsf{Cover}(x) = (x_1, \varrho_1) \cdots (x_k, \varrho_k)$. Define $\varrho = \varrho_k \varrho_{k-1} \dots \varrho_1$, and let $n \in \mathsf{Act}$ and $\varrho' \in T^*$. Then we have:*

$$(\varrho' \prec \varrho \wedge \Lambda(n) \geq \mathsf{Post}_\varphi(\Lambda(x_k), \varrho')) \Rightarrow \beta(x) \leq \alpha(n)$$

Covering Path. Before turning to the proof of Theorem 5, we introduce an additional definition. Our aim is to prove that any reachable state s is covered by some active node. Therefore we define a notion of covering path, which is intuitively a path through active nodes in which each node is labelled with a sequence (a stack) of transitions that remain to be fired to reach a state s' dominating the desired state s. Formally, a covering path is defined as follows:

Definition 9 (Covering Path). *A* covering path *is a sequence $(n_i, \varrho_i)_{i \geq 1} \in (\mathsf{Act} \times T^*)^{\mathbb{N}}$ such that $\Lambda(n_1) \stackrel{\varrho_1}{\Rightarrow}_\varphi \cdot$ and for any $i \geq 1$, we have either*

- *(i) $\varrho_i = \varepsilon$, and then it has no successor, or*
- *(ii) $\varrho_i = t_i \varrho_i'$, then let n be such that $(n_i, t_i, n) \in B$ (possible as $\mathcal{E}$ is T-complete). If $n \in \mathsf{Act}$ then $(n_{i+1}, \varrho_{i+1}) = (n, \varrho_i')$. Otherwise, let $\mathsf{Cover}(n) = (x_1, \eta_1) \cdots (x_k, \eta_k)$, we define $(n_{i+1}, \varrho_{i+1}) = (x_k, \eta_k \dots \eta_1 \cdot \varrho_i')$.*

Note that given a node $n \in \mathsf{Act}$ and $\varrho \in T^*$ such that $\Lambda(n) \stackrel{\varrho}{\Rightarrow}_\varphi \cdot$, there exists a unique covering path $(n_i, \varrho_i)_{i \geq 1}$ such that $(n_1, \varrho_1) = (n, \varrho)$. We say that this path is associated with the pair (n, ϱ).

Example 9 (Example 8 continued). We illustrate the definition of covering path on Example 6. Consider the covering path associated with the pair $(n_1, t_1 t_3 t_4)$. Successor of node n_1 by transition t_1 is the inactive node n_3. We have already shown in Example 8 that $\mathsf{Cover}(n_3) = (n_5, \varepsilon) \cdot (n_6, t_4) \cdot (n_8, \varepsilon)$. In addition, successor of node n_8 by transition t_4 is the active node n_9. Finally, one can verify that the covering path is: $(n_1, t_1 t_3 t_4), (n_8, t_4 t_3 t_4), (n_9, t_3 t_4), (n_8, t_4), (n_9, \varepsilon)$. Note that the marking $\{p_3, p_5\}$ reached from node n_1 by the sequence $t_1 t_3 t_4$ is covered by the marking $\{p_3, \omega p_5\}$ of node n_9. □

Lemma 5. *Let $(n_i, \varrho_i)_{i \geq 1}$ be a covering path. Then we have $\mathsf{Post}_\varphi(\Lambda(n_1), \varrho_1) \leq \mathsf{Post}_\varphi(\Lambda(n_i), \varrho_i)$ for all i. In particular, if for some i we have $\varrho_i = \varepsilon$, we obtain $\mathsf{Post}_\varphi(\Lambda(n_1), \varrho_1) \leq \Lambda(n_i)$.*

Proof. We prove that for any i, we have $\mathsf{Post}_\varphi(\Lambda(n_i), \varrho_i) \leq \mathsf{Post}_\varphi(\Lambda(n_{i+1}), \varrho_{i+1})$. In the definition of covering path, we extend the path only in case (ii). Two cases can occur, in the first one, the property is trivial. In the second one, the property follows from Lemma 4. □

As a consequence of this lemma, to prove the completeness result, it is sufficient to show that for any reachable marking, there exists a finite covering path that covers it. We introduce a notion of cycle for covering paths:

Definition 10. *Let* $(n,t) \in \mathsf{Act} \times T$ *such that* $\Lambda(n) \stackrel{t}{\Rightarrow}_\varphi \cdot$, *and* $(n_i, \varrho_i)_{i\geq 1}$ *be the covering path associated with* (n,t). *The pair* (n,t) *is said to be* singular *if there exists* $i > 1$ *such that* $(n_i, \varrho_i) = (n, t\varrho)$, *with* $\varrho \in T^*$.

Proof of Theorem 5. We will prove that any reachable marking of $(\mathcal{N}, \varphi)$ is covered by some active node. Let $m \in \mathsf{Mark}^\omega_\varphi(P)$ be a reachable marking. There exists $\varrho \in T^*$ such that $m_0 \stackrel{\varrho}{\Rightarrow}_\varphi m$. One can prove (see [10]) that there exists a node $n'_0 \in \mathsf{Act}$ such that $\Lambda(n_0) \leq \Lambda(n'_0)$ (n'_0 covers the root). As a consequence, there exists $m' \in \mathsf{Mark}^\omega_\varphi(P)$ such that $\Lambda(n'_0) \stackrel{\varrho}{\Rightarrow}_\varphi m'$ and $m \leq m'$. We can then consider the covering path associated with the pair (n'_0, ϱ).

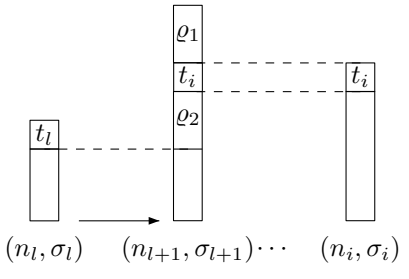


Fig. 5. Stacks of a singular pair

We now prove that all covering paths are finite. This will conclude the proof by Lemma 5. One can prove (see [10]) that if a covering path is infinite, then it contains a singular pair.

We will prove that the MP-exploration $\mathcal{E}$ cannot admit a singular pair. Consider a singular pair $(n,t) \in \mathsf{Act} \times T$, and denote by $(n_i, \sigma_i)_{i\geq 1}$ its infinite covering path. As it is singular, there exists $k > 1$ such that $(n_k, \sigma_k) = (n, t\sigma'_k)$. For any $1 \leq i \leq k$, we write $\sigma_i = t_i\sigma'_i$ (this is possible as the path is infinite and thus never contains empty stacks).

For each $1 < i \leq k$, we define the position $\mathsf{prod}(i) = \max\{1 \leq j < i \mid |\sigma_j| \leq |\sigma_i|\}$. This definition is correct as $|\sigma_1| = |t| = 1$, and for any $1 < i \leq k$, we have $|\sigma_i| \geq 1$ as $\sigma_i \neq \varepsilon$. Intuitively, the value $\mathsf{prod}(i)$ gives the position which is responsible of the addition in the stack of transition t_i. Indeed, let $1 < i \leq k$ and $l = \mathsf{prod}(i)$. As for any position j such that $l < j < i$, we have $|\sigma_j| > |\sigma_i|$, the transition t_i is present in σ_j "at the same height".

Consider now the position $1 < i \leq k$ such that $\alpha(n_i)$ is minimal among $\{\alpha(n_j) \mid 1 \leq j \leq k\}$ (recall that $n_1 = n_k$), and let $l = \mathsf{prod}(i)$. By the T-completeness of $\mathcal{E}$, there exists a node $x \in X$ such that edge (n_l, t_l, x) belongs to B. As we have $|\sigma_l| \leq |\sigma_{l+1}|$, this implies that $x \in \mathsf{Inact}$.

We write the covering function associated with node x as follows: $\mathsf{Cover}(x) = (x_1, \eta_1) \dots (x_k, \eta_k)$. Following the definition of a covering path, we obtain $x_k = n_{l+1}$. In addition, following the above mentionned property of $l = \mathsf{prod}(i)$, there exist two sequences $\varrho_1, \varrho_2 \in T^*$ such that $\eta_k \dots \eta_1 = \varrho_1 t_i \varrho_2$, and verifying:

$$\sigma_{l+1} = \varrho_1 t_i \varrho_2 \sigma'_l \text{ and } \sigma_i = t_i \varrho_2 \sigma'_l$$

This means that the head of the stack σ_l, *i.e.* transition t_l, has been replaced by the sequence $\varrho_1 t_i \varrho_2$, and that between positions $l+1$ and i, transition t_i (which is the head of the stack σ_i), is never consumed. This situation is depicted on Figure 5. In particular, this implies the following property:

$$\Lambda(n_i) \geq \mathsf{Post}_\varphi(\Lambda(n_{l+1}), \varrho_1)$$

Indeed, there are two cases, either $i = l+1$, and then we necessarily have $\varrho_1 = \varepsilon$ and the property is trivial, or $l + 1 < i$, and then we have that the covering path starting in pair (n_{l+1}, ϱ_1) ends in pair (n_i, ε). The result then follows from Lemma 5.

To conclude, we use the key Proposition 4. Indeed, one can verify that the proposition can be applied on nodes x and n_i using sequences $\varrho = \varrho_1 t_i \varrho_2$ and $\varrho' = \varrho_1$. This result yields the following inequality: $\beta(x) \leq \alpha(n_i)$. As x is the successor of n_l by transition t_l, property (ii) of an exploration implies $\alpha(n_l) < \alpha(x)$. As we always have $\alpha(x) \leq \beta(x)$, we finally obtain $\alpha(n_l) < \alpha(n_i)$, which is a contradiction with our choice of i.

5 Comparison and Experiments

We compare MP algorithm with K&M algorithm and with the procedure CoverProc introduced in [7]. This procedure is an alternative for the computation of the MCS. Instead of using a tree structure as in K&M algorithm, it computes a set of pairs of markings, with the meaning that the second marking can be reached from the first one. This is sufficient to apply acceleration. To improve the efficiency, only maximal pairs are stored. Experimental results are presented in Table 1. The K&M and MP algorithms were implemented in Python and tested on a 3 Ghz Xeon computer. The tests set is the one from [7]. We recall in the last column the values obtained for the CoverProc [7] algorithm. Note that the implementation of [7] also was in Python, and the tests were run on the same computer. We report for each test the number of places and transitions of the net and the size of its MCS, the time the K&M and MP algorithms took and the numbers of nodes each algorithm constructed.

As expected the MP algorithm is a lot faster than K&M algorithm and the tree it constructs is, in some instances, dramatically smaller. K&M algorithm

Table 1. K&M and MP algorithm comparison. #P, #T, # MCS : number of places and transitions and size of the MCS of the Petri net. # nodes : number of nodes in the tree constructed by the algorithm.

Test				K&M		MP		CoverProc[7]
name	#P	#T	# MCS	# nodes	time (s)	# nodes	time (s)	time (s)
BasicME	5	4	3	5	< 0.01	5	< 0.01	0.12
Kanban	16	16	1	72226	9.1	114	< 0.01	0.19
Lamport	11	9	14	83	0.02	24	< 0.01	0.17
Manufacturing	13	6	1	81	0.01	30	< 0.01	0.14
Peterson	14	12	20	609	0.2	35	0.02	0.25
Read-write	13	9	41	11139	6.33	76	.06	1.75
Mesh2x2	32	32	256	x	x	6241	18.1	330
Multipool	18	21	220	x	x	2004	4.9	365
pncsacover	31	36	80	x	x	1604	1.6	113
csm	14	13	16	x	x	102	.03	0.34
fms	22	20	24	x	x	809	0.28	2.1

could not compute the MCS for the last five tests (we time out after 20 minutes), while the MP algorithm took less than 20 seconds for all five tests. Note that the time reported for K&M algorithm is the time to build the K&M tree, from this tree one has to extract the minimal coverability set which maybe costly if the set is big (see K&M results in [7]). The MP algorithm directly computes the minimal coverability set (Act), *i.e.* no additional computation is needed.

Regarding CoverProc, the procedure is significantly slower than MP. This can be explained by the fact that considering pairs of markings may increase the number of elements to be stored. Moreover, MP algorithm has, in our view, another important advantage over CoverProc. In MP, the order of exploration is totally free (any exploration strategy yields the MCS) while, to improve the efficiency of the procedure, a depth-first search is applied from markings that have been accelerated.

Acknowledgments. We would like to warmly thank Raymond Devillers, Laurent Doyen, Jean-François Raskin and Olivier De Wolf for fruitful discussions around preliminary versions of this work.

References

1. Finkel, A.: A generalization of the procedure of Karp and Miller to well structured transition system. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 499–508. Springer, Heidelberg (1987)
2. Finkel, A.: The minimal coverability graph for Petri nets. In: Rozenberg, G. (ed.) APN 1993. LNCS, vol. 674, pp. 210–243. Springer, Heidelberg (1993)
3. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part I: Completions. In: Proc. STACS 2009. LIPIcs, vol. 3, pp. 433–444. Leibniz-Zentrum für Informatik (2009)
4. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part II: Complete WSTS. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 188–199. Springer, Heidelberg (2009)
5. Finkel, A., Raskin, J.-F., Samuelides, M., Begin, L.V.: Monotonic extensions of petri nets: Forward and backward search revisited. Electr. Notes Theor. Comput. Sci. 68(6) (2002)
6. Geeraerts, G.: Coverability and Expressiveness Properties of Well-structured Transitions Systems. Thèse de doctorat, Université Libre de Bruxelles, Belgique (2007)
7. Geeraerts, G., Raskin, J.-F., Van Begin, L.: On the efficient computation of the coverability set for petri nets. International Journal of Foundations of Computer Science 21(2), 135–165 (2010)
8. Karp, R.M., Miller, R.E.: Parallel program schemata. Journal of Computer and System Sciences 3(2), 147–195 (1969)
9. Luttge, K.: Zustandsgraphen von Petri-Netzen. Master's thesis, Humboldt-Universitat (1995)
10. Reynier, P.-A., Servais, F.: Minimal Coverability Set for Petri Nets: Karp and Miller Algorithm with Pruning. Research Report 00554919, HAL (2011)

An Algorithm for Direct Construction of Complete Merged Processes

Victor Khomenko and Andrey Mokhov

School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, U.K.
{Victor.Khomenko,Andrey.Mokhov}@ncl.ac.uk

Abstract. *Merged process* is a recently proposed condense representation of a Petri net's behaviour similar to a branching process (unfolding), which copes well not only with concurrency, but also with other sources of state space explosion like sequences of choices. They are by orders of magnitude more condense than traditional unfoldings, and yet can be used for efficient model checking.

However, constructing complete merged processes is difficult, and the only known algorithm is based on building a (potentially much larger) complete unfolding prefix of a Petri net, whose nodes are then merged. Obviously, this significantly reduces their appeal as a representation that can be used for practical model checking.

In this paper we develop an algorithm that avoids constructing the intermediate unfolding prefix, and builds a complete merged process directly. In particular, a challenging problem of truncating a merged process is solved.

Keywords: Merged process, unravelling, Petri net unfolding, model checking, SAT, 2QBF.

1 Introduction

Formal verification, and in particular model checking of concurrent systems is an important and practical way of ensuring their correctness. However, the main drawback of model checking is that it suffers from the *state space explosion* problem [25]. That is, even a relatively small system specification can (and often does) yield a very large state space. To alleviate this problem, many model checking techniques use a condense representation of the full state space of the system. Among them, a prominent technique is McMillan's (finite prefixes of) Petri net unfoldings (see, e.g. [9,12,17]). They rely on the partial order view of concurrent computation, and represent system states implicitly, using an acyclic *unfolding prefix*.

There are several common sources of state space explosion. One of them is concurrency, and the unfolding techniques were primarily designed for efficient verification of highly concurrent systems. Indeed, complete prefixes are often exponentially smaller than the corresponding reachability graphs, because they

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 89–108, 2011.

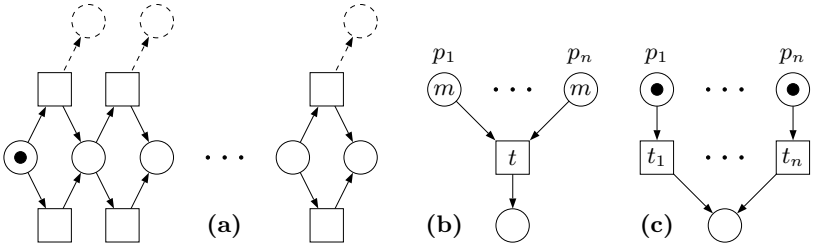


Fig. 1. Examples of Petri nets

represent concurrency directly rather than by multidimensional 'diamonds' as it is done in reachability graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the reachability graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will be isomorphic to the net itself. However, unfoldings do not cope well with some other important sources of state space explosion, in particular with sequences of choices and non-safeness. Below we consider examples illustrating this problem [13].

First, consider Fig. 1(a) with the dashed part not taken into account. The cut-off condition proposed in [9] copes well with this Petri net (since the marking reached after either choice on each stage is the same — in fact, there are few reachable markings), and the resulting prefix is linear in the size of the original Petri net. However, if the dashed part of the figure is added, the smallest complete prefix is exponential, since no event can be declared a cut-off (intuitively, each reachable marking 'remembers' its past). Thus sequences of choices leading to different markings often yield exponential prefixes.

Another problem arises when one tries to unfold non-safe Petri nets, e.g. one in Fig. 1(b). Its smallest complete unfolding prefix contains m^n instances of t, since the standard unfolding *distinguishes between different tokens on the same place.* One way to cope with non-safe nets is to convert them into safe ones and unfold the latter, as was proposed in [9]. However, such an approach destroys concurrency among executed transitions and can lead to very large prefixes; e.g. when applied to the Petri net in Fig. 1(c), it yields an exponential prefix, while the traditional unfolding technique would yield a linear one [9].

The problems with unfolding prefixes described above should be viewed in the light of the fact that all these examples have a very simple structure — viz. they are all acyclic, and thus many model checking techniques, in particular those based on the *marking equation* [12,18,21], could be applied *directly to the original Petri nets.* And so it may happen that an exponential prefix is built for a relatively simple problem!

In [13], a new condense representation of a Petri net's behaviour called *merged processes (MPs)* was proposed, which remedies the problems outlined above. It copes well not only with concurrency, but also with the other mentioned sources of state space explosion, viz. sequence of choices and non-safeness. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for unfoldings can be re-used.

The main idea behind this representation is to fuse some nodes in the complete prefix, and use the resulting net as the basis for verification. For example, the unfolding of the net shown in Fig. 1(a) (even with the dashed part taken into account) will collapse back to the original net after the fusion. In fact, this will happen in all the examples considered above.

It turns out that for a safe Petri net model checking of a reachability-like property (i.e. the existence of a reachable state satisfying a predicate given by a Boolean expression) can be efficiently performed on its MP, and [13] provides a method for reducing this problem to SAT (with the size of the SAT instance being polynomial in the sizes of the MP and the property). Moreover, the experimental results in [13] indicate that this method is quite practical.

However, constructing complete MPs is difficult, and the only known algorithm is based on building a (potentially much larger) complete unfolding prefix, whose nodes are then merged [13]. Obviously, this significantly reduces their appeal as a condense representation that can be used for practical model checking.

In this paper we develop an algorithm that avoids constructing the intermediate unfolding prefix, and builds a complete MP directly. In particular, a challenging problem of truncating an MP is solved, by reducing it to 2QBF, i.e. to satisfiability of a fully quantified Boolean formula of the form $\forall X \exists Y\ \varphi$, where φ is a Boolean formula in conjunctive normal form (CNF) and X and Y are disjoint sets of Boolean variables such that $X \cup Y$ are exactly the variables occurring in φ.

The full version of this paper, including the proofs of all the results, can be found in the technical report [15] (available on the web).

2 Basic Notions

In this section we recall the basic notions concerning SAT, QBF, Petri nets, their unfolding prefixes and merged processes (see also [9,14,17,18,21,23,24,27]).

2.1 Boolean Satisfiability (SAT)

The *Boolean Satisfiability Problem (SAT)* consists in finding a *satisfying assignment*, i.e. a mapping $A : Var_\varphi \to \{0,1\}$ defined on the set of variables Var_φ occurring in a given Boolean expression φ such that $\varphi|_A$ evaluates to 1. This expression is often assumed to be given in the *conjunctive normal form (CNF)* $\varphi = \bigwedge_{i=1}^{n} \bigvee_{l \in L_i} l$, i.e. it is represented as a conjunction of *clauses*, which are disjunctions of *literals*, each literal l being either a variable or the negation of a variable. It is assumed that no two literals in the same clause correspond to the same variable.

SAT is a canonical NP-complete problem. In order to solve it, SAT solvers perform exhaustive search assigning the values 0 or 1 to the variables, using heuristics to reduce the search space [27]. Many leading SAT solvers, e.g. MiniSat [5] and zChaff [20], can be used in the *incremental mode,* i.e. after solving a particular SAT instance the user can slightly change it (e.g. by adding and/or removing a small number of clauses) and execute the solver again. This is often

much more efficient than solving these related instances as independent problems, because on the subsequent runs the solver can use some of the useful information (e.g. learnt clauses [27]) collected so far.

2.2 Quantified Boolean Formulae (QBF)

A *Quantified Boolean Formula* is a formula of the form $Q_1 x_1 \dots Q_n x_n\ \varphi$, where each Q_i is either $\exists$ or $\forall$, each x_i is a Boolean variable, and φ is a Boolean expression. It is *fully quantified* if all the variables occurring in φ are quantified. For convenience, adjacent quantifiers of the same type are often grouped together, so the formula can be re-written as $Q_1 X_1 \dots Q_k X_k\ \varphi$, where X_is are pairwise disjoint non-empty sets of Boolean variables, and the quantifiers' types alternate. Furthermore, φ is often assumed to be given in CNF.

A fully quantified Boolean formula is equal to either 0 or 1, under the natural semantics. The problem of determining if such a formula is equal to 1 is PSPACE-complete. However, if the number of alternations of the quantifiers is a constant k, the complexity is the kth level of Stockmeyer's polynomial hierarchy PH [24]. PH is included in PSPACE, and it is conjectured that PH does not collapse (i.e. the inclusion between its levels is strict); this would imply that the inclusion of PH into PSPACE is also strict.

One can see that checking if a fully quantified QBF instance of the form $\exists X\ \varphi$ is equal to 1 amounts to SAT. Also, QBF instances of the form

$$\forall X \exists Y\ \varphi \tag{1}$$

are of particular interest for this paper. W.l.o.g., one can assume that φ is in CNF, as any instance of (1) can be converted into this form without changing the number of alternation of quantifiers and by only linearly increasing the size of the formula. The problem of checking if (1) equals to 1 is called 2QBF [23]. This problem, though more complicated then SAT (unless PH collapses), is nevertheless much simpler than general QBF and other PSPACE-complete problems like model checking (as these problems are separated by infinitely many levels of PH, unless PH collapses). Furthermore, though general QBF solvers have not reached maturity level of contemporary SAT solvers, 2QBF admits specialised methods, e.g. based on a pair of communicating SAT solvers [23].

2.3 Petri Nets

A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, F)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. The net is *finite* if both P and T are finite sets.

A *marking* of N is a multiset M of places, i.e. $M : P \to \mathbb{N}$. We adopt the standard rules about drawing nets, viz. places are represented as circles, transitions as boxes, the flow relation by arcs, and the marking is shown by placing tokens within circles. As usual, ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid (y,z) \in F\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid (z,y) \in F\}$ denote the *preset* and *postset* of $z \in P \cup T$. In this paper, the presets of transitions are restricted to be non-empty, i.e. ${}^\bullet t \neq \emptyset$ for every $t \in T$. For a finite net N, we define the *size* of N as $|N| \stackrel{\text{df}}{=} |P| + |T| + |F|$.

A *Petri net (PN)* is a pair $\Sigma \stackrel{\text{df}}{=} (N, M_\Sigma)$ comprising a finite net $N = (P, T, F)$ and an (initial) marking M_Σ. A transition $t \in T$ is *enabled* at a marking M, denoted $M[t\rangle$, if for every $p \in {}^\bullet t$, $M(p) \geq 1$. Such a transition can be *executed* or *fired*, leading to a marking M' given by $M' \stackrel{\text{df}}{=} M - {}^\bullet t + t^\bullet$. We denote this by $M[t\rangle M'$. The set of *reachable* markings of Σ is the smallest (w.r.t. $\subset$) set $[M_\Sigma\rangle$ containing M_Σ and such that if $M \in [M_\Sigma\rangle$ and $M[t\rangle M'$ for some $t \in T$ then $M' \in [M_\Sigma\rangle$. For a finite sequence of transitions $\sigma = t_1 \ldots t_k$ $(k \geq 0)$, we write $M[\sigma\rangle M'$ if there are markings $M_1, \ldots, M_{k+1}$ such that $M_1 = M$, $M_{k+1} = M'$ and $M_i[t_i\rangle M_{i+1}$, for $i = 1, \ldots, k$. If $M = M_\Sigma$, we call σ an *execution* of Σ.

A marking is *deadlocked* if it does not enable any transitions. A PN Σ is *deadlock-free* if none of its reachable marking is deadlocked. It 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, Σ is *bounded* if it is k-bounded for some $k \in \mathbb{N}$. One can show that $[M_\Sigma\rangle$ is finite iff Σ is bounded.

A powerful tool in analysis of PNs is the so called *marking equation* [21], which states that the final number of tokens in any place p of a PN Σ can be calculated as $M_\Sigma(p)$ plus the number of tokens brought to p minus the number of tokens taken from p. The feasibility of this equation is a necessary condition for a marking to be reachable from M_Σ. However, the marking equation can have *spurious* solutions which do not correspond to any execution of Σ, i.e. it is not a sufficient condition, and yields an over-approximation of $[M_\Sigma\rangle$. However, for some PN classes, in particular for acyclic PNs (which include branching processes defined below), this equation provides an *exact* characterisation of $[M_\Sigma\rangle$ [21, Th. 16] — providing the basis for model checking algorithms based on unfolding prefixes [12,18]. Moreover, exact characterisations of $[M_\Sigma\rangle$ can be obtained for some further PN classes by augmenting the marking equation with additional constraints; in particular, such a characterisation was obtained in [13] for MPs, and it will be essential for the proposed unravelling algorithm.

2.4 Branching Processes

A *branching process* β of a PN Σ is a finite or infinite labelled acyclic net which can be obtained through unfolding Σ, by successive firings of transitions, under the following assumptions: (i) one starts from a set of places (called *conditions*), one for each token of the initial marking; (ii) for each new firing a fresh transition (called an *event*) is generated; and (iii) for each newly produced token a fresh place (also called a *condition*) is generated. Each event (resp. condition) is labelled by the corresponding transition (resp. place on which the corresponding token was present).

There exists a unique (up to isomorphism) maximal (w.r.t. the prefix relation) branching process β_Σ of Σ called the *unfolding* of Σ [6,9]. For example, the unfolding of the PN in Fig. 2(a) is shown in part (b) of this figure.

The unfolding β_Σ is infinite whenever Σ has executions of unbounded length; however, if Σ has finitely many reachable states then the unfolding eventually starts to repeat itself and thus can be truncated (by identifying a set of *cut-off* events beyond which the unfolding procedure is not continued) without loss of

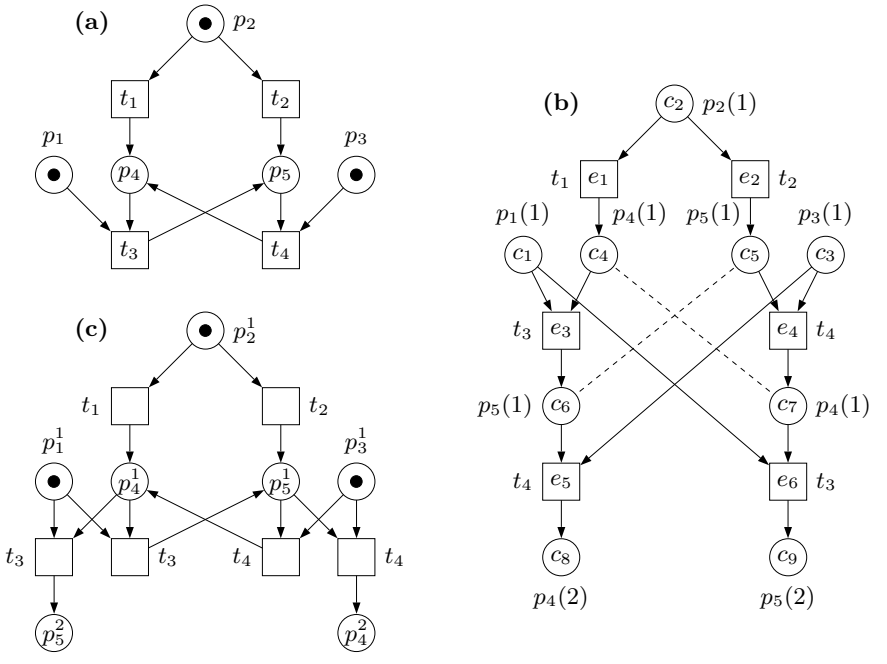


Fig. 2. A Petri net **(a)**; its unfolding with the occurrence-depths of conditions shown in brackets and the conditions to be fused connected by dashed lines **(b)**; and its unravelling **(c)**

essential information. For a branching process β obtained in this way, the sets of conditions, events, arcs and cut-off events of β will be denoted by B, E, G and E_{cut}, respectively (note that $E_{cut} \subseteq E$), and the labelling function by h. Note that when talking about an *execution* of β, we will mean any execution from its implicit initial marking M_β that comprises the initial conditions.

Since β is acyclic, the transitive closure of its flow relation is a partial order $<$ on $B \cup E$, called the *causality relation*. (The reflexive order corresponding to $<$ will be denoted by $\leq$.) Intuitively, all the events which are smaller than an event $e \in E$ w.r.t. $<$ must precede e in any run of β containing e.

Two nodes $x, y \in B \cup E$ are in *conflict*, denoted $x \# y$, if there are distinct events $e, f \in E$ such that ${}^\bullet e \cap {}^\bullet f \neq \emptyset$ and $e \leq x$ and $f \leq y$. Intuitively, no execution of β can contain two events in conflict. Two nodes $x, y \in B \cup E$ are *concurrent*, denoted $x \parallel y$, if neither $x \# y$ nor $x \leq y$ nor $y \leq x$. Intuitively, two concurrent events can be enabled simultaneously, and executed in any order, or even concurrently, and two concurrent conditions can be simultaneously marked. For example, in the branching process shown in Fig. 2(b) the following relationships hold: $e_1 < e_5$, $e_3 \# e_4$ and $c_1 \parallel c_4$.

Due to structural properties of branching processes (such as acyclicity), the reachable markings of Σ can be represented using *configurations* of β. A *configuration* is a finite set of events $C \subseteq E$ such that (i) for all $e, f \in C$, $\neg(e \# f)$;

and (ii) for every $e \in C$, $f < e$ implies $f \in C$. For example, in the branching process shown in Fig. 2(b) $\{e_1, e_3, e_5\}$ is a configuration whereas $\{e_1, e_2, e_3\}$ and $\{e_1, e_5\}$ are not (the former includes events in conflict, $e_1 \# e_2$, while the latter does not include e_3, a causal predecessor of e_5). Intuitively, a configuration is a partially ordered execution, i.e. an execution where the order of firing of some of its events (viz. concurrent ones) is not important. For every event e of β_Σ, $[e] \stackrel{\text{df}}{=} \{f \mid f \text{ is an event of } \beta_\Sigma \text{ and } f \leq e\}$ is called the *local configuration* of e. Intuitively, it comprises e and all its causal predecessors.

After starting β from the implicit initial marking M_β and executing all the events in C, one reaches the marking (of β) denoted by $Cut(C)$. $Mark(C) \stackrel{\text{df}}{=} h(Cut(C))$ denotes the corresponding marking of Σ, reached by firing a transition sequence corresponding to the events in C. Let E_{cut} be a set of events of β. Then β is *marking-complete w.r.t.* E_{cut} if, for every reachable marking M of Σ, there is a configuration C of β such that $C \cap E_{cut} = \emptyset$ and $Mark(C) = M$. Moreover, β is *complete* if it is marking-complete and, for each configuration C of β such that $C \cap E_{cut} = \emptyset$ and for each event $e \notin C$ of β_Σ such that $C \cup \{e\}$ is a configuration of β_Σ, e is in β. This *preservation of firings* property is useful for deadlock detection.

Complete branching processes are often called complete (unfolding) *prefixes*. One can build a complete prefix in such a way that the number of non-cut-off events $|E \setminus E_{cut}|$ does not exceed the number of reachable markings of Σ [9,12]. The unfolding algorithms described there declare an event e cut-off if there is a smaller (w.r.t. some well-founded partial order $\lhd$, called an *adequate order*) *corresponding configuration* C in the already built part of the prefix containing no cut-off events and such that $Mark([e]) = Mark(C)$. With some natural conditions on the way this cutting is performed, the resulting prefix is unique, even though the unfolding algorithm may be non-deterministic. This unique prefix is called *canonical,* and it can be defined in an algorithm-independent way [14].

As already mentioned, in acyclic nets like β, a marking is reachable iff the corresponding marking equation has a solution; since there is a correspondence between the reachable markings of Σ and those of any of its marking-complete branching processes β, the latter can be used for efficient model checking [12,16,17,18].

2.5 Merged Processes

Let β be a branching process of a PN Σ, and x be one of its nodes (condition or event). The *occurrence-depth* of x is defined as the maximum number of $h(x)$-labelled nodes on any directed path starting at an initial condition and terminating at x in the acyclic digraph representing β. The occurrence-depth is well-defined since there is always at least one such a path, and the number of all such paths is finite. In Fig. 2(b) the occurrence-depths of conditions are shown in brackets.

Definition 1 (merged process). *Given a branching process β, the corresponding* merged process $\mu = \mathfrak{Merge}(\beta)$ *is a PN which is obtained in two steps, as follows:*

Step 1: *the places of μ, called* mp-conditions, *are obtained by fusing together all the conditions of β which have the same labels and occurrence-depths; each mp-condition inherits its label and arcs from the fused conditions, and its initial marking is the total number of the initial conditions which were fused into it.*
Step 2: *the transitions of μ, called* mp-events, *are obtained by merging all the events which have the same labels, presets and postsets (after Step 1 was performed); each mp-event inherits its label from the merged events (and has exactly the same connectivity as either of them), and it is a* cut-off mp-event *iff* <u>*all*</u> *the events merged into it were cut-off events in β.*

Moreover, $\mu_\Sigma \stackrel{\text{df}}{=} \mathfrak{Merge}(\beta_\Sigma)$ is the merged process corresponding to the unfolding of Σ, called the unravelling *of Σ.* ◊

Fig. 2(b,c) illustrates this definition, which also yields an algorithm for computing $\mathfrak{Merge}$. In the sequel, $\hbar$ will denote the mapping of the nodes of β to the corresponding nodes of μ, and we will use 'hats' and 'mp-' to distinguish the elements of μ from those of β, in particular, $\widehat{E}$, $\widehat{B}$, $\widehat{G}$, $\widehat{M_\mu}$, $\widehat{E_{cut}}$ will denote the set of mp-events, the set of mp-conditions, the flow relation, the initial marking and the set of cut-off mp-events of μ, and $\widehat{h}$ will denote the mapping of the nodes of μ to the corresponding nodes of Σ.

Note that in general, μ is not acyclic (cycles can arise due to criss-cross fusions of conditions, as illustrated in Fig. 2(b,c)). This, in turn, leads to complications for model checking; in particular, the marking equation for μ can have spurious solutions not corresponding to any reachable marking, and thus has to be augmented with additional constraints, see Sect. 2.6.

A multiset $\widehat{C}$ of mp-events is an *mp-configuration* of μ if $\widehat{C} = \hbar(C)$ for some configuration C of β_Σ. Note that there is a subtlety in this definition: we have to use the unfolding β_Σ of Σ rather than an arbitrary branching process β satisfying $\mu = \mathfrak{Merge}(\beta)$, since μ may contain mp-configurations which are not $\hbar$-images of any configuration in such a β, i.e. the mp-configurations of μ might be ill-defined if μ can arise from several different branching processes. E.g., for the PN in Fig. 3(a), consider the unfolding β_Σ shown in part (b) and the branching process β shown in part (c) of this figure: both give rise to the same (up to isomorphism) MP μ shown in part (d) of the figure, and the mp-configuration $\{\widehat{e}_2, \widehat{e}_3\}$ of μ is not an image of any configuration of β, but it is the image of the configuration $\{e_2, e_4\}$ of β_Σ.

If $\widehat{C}$ is an mp-configuration then the corresponding *mp-cut* $Cut(\widehat{C})$ is defined as the marking of μ reached by executing all[1] the events of $\widehat{C}$ starting from the initial marking $\widehat{M_\mu}$. Moreover, $Mark(\widehat{C}) \stackrel{\text{df}}{=} \widehat{h}(Cut(\widehat{C}))$. Note that if $\widehat{C} = \hbar(C)$ then $Mark(\widehat{C}) = Mark(C)$.

MPs of safe PNs have a number of special properties. First of all, their mp-configurations are sets (rather than multisets) of mp-events. Moreover, there is a one-to-one correspondence between the mp-configurations of the unravelling

[1] I.e., each mp-event in $\widehat{C}$ is executed as many times as it occurs in $\widehat{C}$, and no other event is executed — this is always possible. $Cut(\widehat{C})$ can be efficiently computed using, e.g. the marking equation [21].

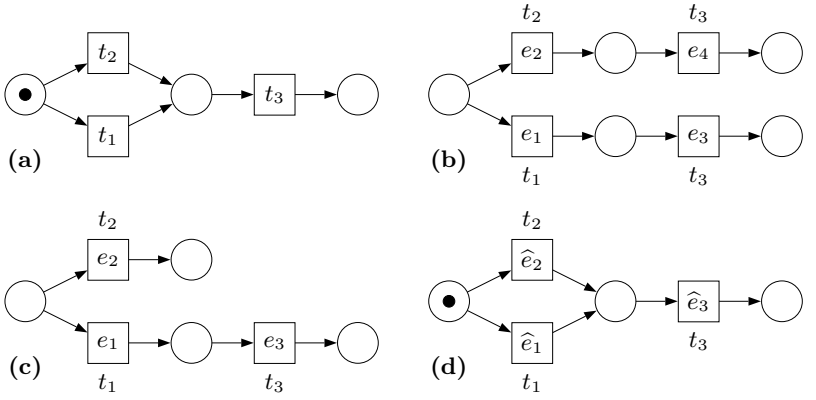


Fig. 3. A Petri net **(a)**; its unfolding **(b)**; one of its unfolding prefixes **(c)**; and the merged process corresponding to both the unfolding and the depicted prefix **(d)**

and the configurations of the unfolding, such that each mp-configuration $\widehat{C}$ is isomorphic to the corresponding configuration C, in the sense that the $\widehat{h}$-labelled digraph induced by the nodes in $\widehat{C} \cup \widehat{C}^\bullet \cup \widehat{M_\mu}$ in the unravelling is isomorphic to the h-labelled digraph induced by the nodes in $C \cup C^\bullet \cup M_\beta$ in the unfolding. Furthermore, one can partially transfer the notion of an event's local configuration from branching processes to MPs. Though the notion of *the* local configuration of an mp-event $\widehat{e}$ does not make sense, as $\widehat{e}$ can have many of them, one can specify some additional information to make this local configuration unique, viz. an mp-configuration $\widehat{C}$ such that $\widehat{e} \in \widehat{C}$ within which the local configuration of $\widehat{e}$ must be contained; it will be denoted by $[\widehat{e}]_{\widehat{C}}$. (Note that the uniqueness is still not guaranteed for MPs of unsafe PNs.)

Canonical merged processes. Since $\mathfrak{Merge}$ is a deterministic transformation, one can define the *canonical* MP as $\mathfrak{Merge}(\beta)$, where β is the canonical unfolding prefix of [14]. This allows for an easy import of the results of [14] related to the canonicity. Furthermore, it simplifies the proof of correctness of the algorithm proposed in this paper, as it is enough to prove that it constructs $\mathfrak{Merge}(\beta')$, where β' is obtained by removing the cut-off events from the canonical prefix.

However, $\mathfrak{Merge}$ has problems with the cut-offs, as when merging events, it loses the information about their cut-off status, see Def. 1. For example, in Fig. 3(b) e_4 can be declared a cut-off, but this information is lost when it is merged with a non-cut-off event e_3, resulting in the MP shown in part (d) of the figure. This causes problems with the notion of completeness, which cannot be easily fixed; hence [13] suggests to be content with marking-completeness, which is sufficient for model checking.

The MPs produced by the unravelling algorithm proposed in this paper do not have this problem, and can be easily made complete. (And they coincide with the MPs produced by $\mathfrak{Merge}$ on non-cut-off mp-events.) Therefore, they may be considered as better candidates for being called canonical.

Finiteness of merged processes. It is easy to show that $\mathfrak{Merge}(\beta)$ is finite iff β is finite [13]. Again, this allows one to import all the finiteness results proved for unfolding prefixes [9,14].

Completeness of merged processes. Marking-completeness of MPs is defined similarly to that of branching processes. An MP μ is *marking-complete w.r.t.* $\widehat{E_{cut}}$ if, for every reachable marking M of Σ, there exists an mp-configuration $\widehat{C}$ of μ such that $\widehat{C} \cap \widehat{E_{cut}} = \emptyset$ and $Mark(\widehat{C}) = M$. Moreover, μ is *complete* if it is marking-complete and, for each mp-configuration $\widehat{C}$ of μ such that $\widehat{C} \cap E_{cut} = \emptyset$ and each mp-event $\widehat{e} \notin \widehat{C}$ of μ_Σ such that $\widehat{C} \cup \{\widehat{e}\}$ is an mp-configuration of μ_Σ, $\widehat{e}$ is in μ (i.e. the firings are preserved).

Let C be a configuration of β and $\widehat{C} = \hbar(C)$ be the corresponding mp-configuration of μ. One can easily show that if C contains no cut-off events then $\widehat{C}$ contains no cut-off mp-events, and that $Mark(C) = Mark(\widehat{C})$. Hence, if β is a marking-complete branching process then $\mathfrak{Merge}(\beta)$ is a marking-complete MP [13].

Unfortunately, no such result holds for full completeness: [13] gives an example where $\mathfrak{Merge}(\beta)$ contains a false deadlock, even though β is a complete prefix of a deadlock-free PN. Hence, model checking algorithms developed for unfolding prefixes relying on the preservation of firings (e.g. some of the deadlock checking algorithms in [11,12,16,17,18]) cannot be easily transferred to MPs. However, marking-completeness is sufficient for most purposes, as the transitions enabled by the final state of an mp-configuration can be easily found using the original PN. (The model checking approach of [13], recalled in in Sect. 2.6, does not rely on preservation of firings.)

In contrast, the algorithm proposed in this paper allows one to build a complete MP (i.e. to preserve firings), and hence import the model checking algorithms making use of cut-off events, in particular those for deadlock detection.

The size of a merged process. The fusion of conditions in Def. 1 can only decrease the number of conditions, without affecting the number of events or arcs; moreover, merging events can only decrease the number of events and arcs, without affecting the number of conditions. Hence, $|\mathfrak{Merge}(\beta)| \leq |\beta|$, i.e. MPs are no less condense than branching processes (but can be exponentially smaller, see the examples in Sect. 1). This allows one to import all the upper bounds proved for unfolding prefixes [9,14]. In particular, since for every safe PN Σ one can build a marking-complete branching process with the number of events not exceeding the number of reachable markings of Σ, the corresponding MP has the same upper bound on the number of its events.

However, the upper bound given by the size of the unfolding prefix is rather pessimistic; in practice, MPs turn out to be much more compact than the corresponding unfolding prefixes. Fig. 4 compares the sizes of the original PNs, their complete unfolding prefixes and the corresponding MPs [13] on some of the benchmarks collected by J.C. Corbett [4]. The chart on the left shows that MPs are usually much smaller than unfoldings. The picture on the right does not give the sizes of the unfolding prefixes in order to improve the scale; it shows

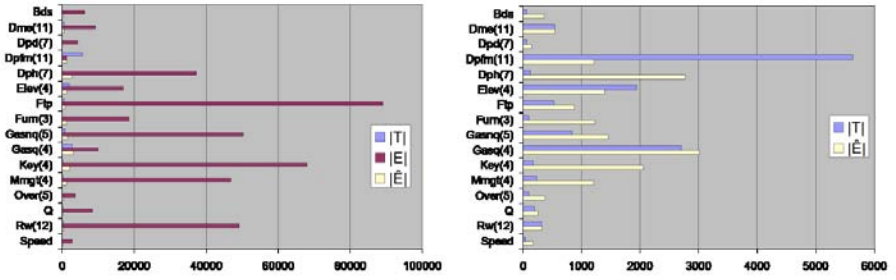


Fig. 4. The numbers of transitions in the original PN, events in its complete prefix, and mp-events in the corresponding MP (left); the numbers of transitions in the original PNs and mp-events in the corresponding MP (right)

that in most cases the sizes of MPs are close to those of the original PNs (and even smaller in some cases due to the presence of dead nodes in the original PN).

Since MPs are inherently more compact than unfolding prefixes, it would be natural to seek sharper upper bounds than the trivial one given by the size of the unfolding prefix. In particular, [13] identifies two subclasses of PNs whose smallest complete unfolding prefixes can be exponential in the size of the original PN, but whose complete MPs are only polynomial:

- Acyclic PNs (the unravelling of such a net Σ coincides with the net obtained from Σ by removing its dead transitions and unreachable places).
- Live and safe free-choice PNs with transitions' postsets of bounded size (the degree of the polynomial depends on the bound on transitions' postsets sizes). Note that the expressive power of this class of PNs is comparable with that of the full class of live and safe free-choice PNs, since every transition with a large postset can be replaced by a tree of transitions with postsets of bounded size, so that the behaviour of the PN is preserved.

2.6 Model Checking Based on Merged Processes

We briefly recall the main ideas of the reduction of a reachability-like property R to SAT using MPs (see [13] for more details). Given a marking-complete MP μ of a safe PN, we associate with each non-cut-off mp-event $\widehat{e}$ of μ a Boolean variable $\mathsf{conf}_{\widehat{e}}$. Hence, every assignment A to these variables corresponds to the set $\widehat{C} \stackrel{\text{df}}{=} \{\widehat{e} \mid A(\mathsf{conf}_{\widehat{e}}) = 1\}$ of mp-events of μ. The SAT instance has the form $\mathcal{CONF} \wedge \mathcal{VIOL}$, where the role of the *configuration constraint,* $\mathcal{CONF}$, is to ensure that $\widehat{C}$ is an mp-configuration of μ (not just an arbitrary set of mp-events), and the role of the *violation constraint,* $\mathcal{VIOL}$, is to express that the property R holds for the marking $\mathit{Mark}(\widehat{C})$.

$\mathcal{CONF}$ has the form $\mathcal{ME} \wedge \mathcal{ACYCLIC} \wedge \mathcal{NG}$, where $\mathcal{ME}$ expresses that $\widehat{C}$ is a solution of the marking equation [21] (i.e. the mp-events in $\widehat{C}$ provide a valid token flow), $\mathcal{ACYCLIC}$ conveys that the digraph induced by $\widehat{M_\mu} \cup \widehat{C} \cup \widehat{C}^\bullet$ is acyclic, and $\mathcal{NG}$ (no-gap) expresses that the mp-conditions labelled by the same place p are visited in an order consistent with their occurrence-depths, without gaps.

To construct $\mathcal{VIOL}$ from R, one can first build a Boolean formula 'computing' $Mark(\widehat{C})$, i.e. relating the variables conf_* with new variables mark_p, for each place p of the PN, tracing whether $p \in Mark(\widehat{C})$. Intuitively, $\mathsf{mark}_p = 1$ iff some mp-condition $\widehat{c}$ labelled by p is in $Cut(\widehat{C})$, i.e. $\mathsf{conf}_{\widehat{e}} = 1$ for some $\widehat{e} \in {}^\bullet\widehat{c}$ and $\mathsf{conf}_{\widehat{e}} = 0$ for all $\widehat{e} \in \widehat{c}^\bullet$. Then one can simply substitute all the references to places in R by the corresponding variables mark_*, which yields the $\mathcal{VIOL}$ constraint.

The existence of a reduction to SAT means that MPs are much more amenable to model checking than general safe PNs — e.g. most of 'interesting' behaviourial properties are known to be PSPACE-complete for safe PNs [7], whereas checking reachability-like properties on a marking-complete MP is only NP-complete. Since many such properties are known to be NP-complete already for unfolding prefixes, the theoretical complexity is not worsened if one uses MPs instead of unfolding prefixes.

On the practical side, the SAT instances for MPs are more complicated than those for unfolding prefixes. However, as MPs are usually much smaller than unfolding prefixes, the model checking runtimes are quite similar according to the experiments in [13]. Since space considerations are of utmost importance in model checking, these results can be regarded as positive. Furthermore, the encoding used in [13] for the $\mathcal{ACYCLIC}$ constraint was rather inefficient, and there is a chance that it can be significantly improved, speeding up model checking.

3 Unravelling Algorithm

In this section we present the main contribution of this paper, viz. an algorithm for constructing a complete MP of a safe PN, that avoids building a complete unfolding prefix. The algorithm is shown in Fig. 5, and we state the results concerning its correctness; their proofs can be found in the technical report [15].

The algorithm constructs the MP by starting from the initial mp-conditions (they exactly correspond to the initially marked places of the PN), and repeatedly adding *possible extensions* to it, until a certain termination condition is fulfilled.

Definition 2 (Possible extension). *An mp-event $\widehat{e}$ of μ_Σ is a* possible extension *of an MP μ of Σ if $\widehat{e}$ is not in μ and adding $\widehat{e}$ (together with the mp-conditions in $\widehat{e}^\bullet$ that are not in μ yet) to μ results in an MP of Σ.*

One can compute the set of possible extensions of μ as follows. For each transition t of Σ, one looks for an mp-configuration $\widehat{C}$ of μ which can be extended by an instance $\widehat{e}$ of t that is not in μ yet, such that $\widehat{C} \cup \{\widehat{e}\}$ is an mp-configuration of μ_Σ. This can be formulated as a model checking problem and reduced to SAT as explained in Sect. 2.6. Once a possible extension is computed, a constraint preventing its re-computation is added to the SAT instance, and the problem is solved again, until it becomes unsatisfiable. The process terminates when all possible extensions corresponding to every transition t are computed. Note that the SAT instances to be solved are very similar, and incremental SAT can be

input : Σ — a safe PN
output : μ — a marking-complete or complete MP of Σ

$\mu \leftarrow$ the branching process comprised of the initial mp-conditions
$conf\_sz \leftarrow 0$ /* current configuration size */
repeat
  $conf\_sz \leftarrow conf\_sz + 1$
  $pe \leftarrow \{$possible extensions of $\mu\}$ /* SAT */
  $cand \leftarrow \{\widehat{e} \in pe \mid \widehat{e}$ has a local configuration of size $conf\_sz$ in $\mu\}$ /* SAT */
  /* filter out potential cut-offs */
  $slice \leftarrow \{\widehat{e} \in cand \mid \neg$MaybeCutOff$(\mu \oplus cand, \widehat{e}, conf_sz)\}$
  $\mu \leftarrow \mu \oplus slice$
  /* **Invariant:** $\mu = \mathfrak{Merge}(\beta_{\lceil conf\_sz \rceil})$ */
until $slice = \emptyset \wedge$
  $\neg\exists \widehat{e} \in pe : \widehat{e}$ has a local mp-configuration of size $> conf\_sz$ in $\mu \oplus pe$ /* SAT */
/* optionally assign $\mu \leftarrow \mu \oplus pe$ and mark the mp-events in pe as cut-offs */

/* Check if each local mp-configuration of $\widehat{e}$ of size $conf\_sz$ in $\overline{\mu}$ contains a cut-off */
MaybeCutOff$(\overline{\mu}, \widehat{e}, conf_sz) \equiv$
  /* 2QBF */
  $\forall$ local mp-configurations $\widehat{C}$ of $\widehat{e}$ in $\overline{\mu}$ such that $|\widehat{C}| = conf\_sz$:
    $\exists \widehat{f} \in \widehat{C} : \exists$ mp-configuration $\widehat{C}'$ in $\overline{\mu} : Mark([\widehat{f}]_{\widehat{C}}) = Mark(\widehat{C}') \wedge [\widehat{f}]_{\widehat{C}} \lhd \widehat{C}'$

Fig. 5. An unravelling algorithm that avoids constructing a complete unfolding prefix. It assumes that the adequate order $\lhd$ refines the size order.

used to optimise the whole process. Given an MP μ and a set S of possible extensions of μ, we denote by $\mu \oplus S$ the MP obtained by adding the mp-events in S to μ, together with the mp-conditions in their postsets that are not in μ yet.

The cut-off check is implemented using the MaybeCutOff predicate. It naturally translates to a 2QBF instance, and a 2QBF solver is used to check if it holds. Note that until the algorithm terminates, it is not possible to designate an mp-event $\widehat{e}$ as a cut-off, as $\widehat{e}$ can have many local mp-configurations not all of which are known yet (as adding other mp-events to μ can result in $\widehat{e}$ acquiring new local mp-configurations). However, all the local mp-configurations of $\widehat{e}$ of size up to $conf\_sz$ are guaranteed to be in μ (except ones containing a cut-off), and if the adequate order[2] $\lhd$ refines the size order, it is guaranteed that if MaybeCutOff does not hold for $\widehat{e}$ (and so $\widehat{e}$ is added to μ), it will never hold for $\widehat{e}$ in future, i.e. an added mp-event will not suddenly become cut-off after further mp-events are added to μ.

To summarise, the algorithm adds possible extensions to the MP being constructed in rounds, where in round k a possible extension is considered for

[2] Since for a safe PN Σ configurations of β_Σ are isomorphic to the corresponding mp-configurations of μ_Σ, the adequate orders used for truncating the unfolding, see e.g. [9], can be re-used in the context of MPs.

addition if it has at least one local configuration of size k that contains no cut-offs. In this way, in k th round the algorithm can already perform a one-sided check of the MaybeCutOff condition for local configurations of size k, and if it does not hold then it *definitely* knows that the mp-event cannot be a cut-off[3] and so can be added to the MP. Hence the same mp-event can be considered for addition several times (in different rounds), and each time more of its local configurations are taken into account. Eventually the termination criterion becomes satisfied, which ensures that all the possible extensions remaining in *pe* can be declared a cut-off, and the algorithm stops.

In general, if β is a complete branching prefix, $\mathfrak{Merge}(\beta)$ can still be incomplete (only marking-completeness is guaranteed). In contrast, the proposed algorithm can construct a complete MP (i.e. preserve firings in addition to marking-completeness). This is achieved by adding the possible extensions that are still in *pe* to μ and marking them as cut-offs in the optional operator following the main loop. This incurs almost no overhead, but allows one to employ more efficient model checking algorithms, in particular for deadlock detection.

3.1 Correctness of the Algorithm

We assume that the adequate order $\lhd$ refines the size order on mp-configurations, i.e. $|\widehat{C'}| < |\widehat{C''}|$ implies $\widehat{C'} \lhd \widehat{C''}$ (this is the case for most adequate orders proposed in literature). We will denote by $\beta^{\lhd}$ the unfolding prefix obtained by removing the cut-off events from the canonical unfolding prefix of Σ that was built using $\lhd$ as the adequate order, with all configurations (not only the local ones, as often the case in unfoldings) being allowed as cut-off correspondents. Furthermore, $\beta^{\lhd}_{\lceil n \rceil}$ will denote the prefix obtained from $\beta^{\lhd}$ by removing all the events with local configurations of size exceeding n.

The result below proves the crucial invariant of the algorithm.

Proposition 1 (Invariant). *After each execution of the body of the main loop the invariant* $\mu = \mathfrak{Merge}(\beta^{\lhd}_{\lceil conf\_sz \rceil})$ *holds.*

The soundness of the algorithm trivially follows from the invariant of Prop. 1, and so does not require a separate proof.

Proposition 2 (Soundness). *If the algorithm adds an mp-event* $\widehat{e}$ *to the merged process being constructed then there is an event* e *in* $\beta^{\lhd}$ *such that* $\hbar(e) = \widehat{e}$.

The following results state that the generated merged process is marking-complete (and even complete if the optional part of the algorithm is performed), and that the algorithm terminates.

Proposition 3 (Completeness). *Upon termination of the algorithm the resulting merged process* μ *is marking-complete. Moreover, it is complete if the optional statement after the main loop is executed.*

Proposition 4 (Termination). *The algorithm terminates.*

[3] The opposite is not true, i.e. if MaybeCutOff predicate holds for some mp-event, it cannot be immediately declared a cut-off, as it might acquire new mp-configurations in future.

3.2 Optimisations

The algorithm in Fig. 5 is formulated in such a way that it would be easier to prove its correctness. In practice, however, a number of optimisations can be introduced to improve the performance.

Computing possible extensions. The algorithm does not have to re-compute the set of possible extensions from scratch on every iteration of the main loop. It is sufficient to update the set *pe* left from the previous iteration, by removing the events in *slice* from it (as they have been added to μ) and adding only those possible extensions that are not in μ or *pe* yet.

Cut-off check. If for a possible extension $\widehat{e}$, there is an mp-condition $\widehat{c} \in \widehat{e}^\bullet$ such that there is no mp-condition labelled by $\widehat{h}(\widehat{c})$ in $\mu \oplus (cand \setminus \{\widehat{e}\})$, the predicate MaybeCutOff$(\mu \oplus cand, \widehat{e}, conf_sz)$ cannot hold, and so this check becomes unnecessary.

Counterexamples in the 2QBF solver. If the 2QBF solver is implemented by a pair of SAT solvers [23], one can reduce the number of communication rounds by ensuring that the counterexamples generated by the auxiliary solver are mp-configurations of size $conf\_sz$. This can be achieved by initialising the auxiliary solver with the formula expressing this condition, rather than with an empty formula.

4 Experimental Results

We have evaluated a prototype implementation of our algorithm on a popular set of benchmarks collected by J.C. Corbett [4]. The same benchmarks were used in [13]; however, there are some important differences in our setup, which resulted in different figures in the tables, in particular the sizes of MPs and unfolding prefixes.

The first major difference is that the size-lexicographical adequate order [9] on configurations was used for producing cut-offs. For consistency, the same order is used to produce the unfolding prefixes. This order is not total (though it is not far from being total), and is inferior to the ERV total adequate order proposed in [9]. However, the latter is difficult to encode as a Boolean formula, and our early experiments indicated that it would dominate the size of the SAT instance and increase the satisfiability checking time by an order of magnitude. Hence, we are currently looking at other adequate orders, in particular, we believe that a Boolean encoding of an order similar to the $\prec_{sl}^{d}$ order [8] would be much better; this, however, is left for future research.

The second important difference is that the proposed algorithm can use any configuration as a cut-off correspondent, unlike the standard unfolding technique that uses only local ones. This is due to a subtle point in the correctness proof of our algorithm, which currently does not allow one to restrict the cut-off check to local configurations; moreover, it is not clear that this would result in efficiency gains anyway. This is in contrast to the standard unfolding technique, where restricting the cut-off check in such a way significantly reduces its complexity

Table 1. Experimental results

Benchmark	Net		Unfolding prefix				Merged process			
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	t[s]	$\|\widehat{B}\|$	$\|\widehat{E}\|$	$\|\widehat{E}_{cut}\|$	t[s]
ABP	43	95	337	167	56	<1	73	79	23	3
BDS	53	59	38539	19629	10218	2	88	108	55	6
BYZ	504	409	908702	330906	18944	11081	651	328	7	273
FTP	176	529	249163	124655	50035	271	260	666	185	1620
Q	163	194	21867	11406	1621	2	245	248	36	2256
SPEED	33	39	9023	5380	1910	<1	91	178	52	19
CYCLIC(6)	47	35	112	50	7	<1	84	45	7	<1
CYCLIC(9)	71	53	172	77	10	<1	129	69	10	1
CYCLIC(12)	95	71	232	104	13	<1	174	93	13	2
DAC(9)	63	52	167	95	0	<1	63	52	8	<1
DAC(12)	84	70	260	146	0	<1	84	70	11	<1
DAC(15)	105	88	371	206	0	<1	105	88	14	<1
DP(8)	48	32	368	176	56	<1	80	48	16	1
DP(10)	60	40	580	280	90	<1	100	60	20	2
DP(12)	72	48	840	408	132	<1	120	72	24	3
DPD(5)	45	45	1582	790	211	<1	71	67	29	1
DPD(6)	54	54	3786	1892	499	<1	85	81	35	2
DPD(7)	63	63	8630	4314	1129	<1	99	95	41	4
DPFM(5)	27	41	67	31	20	<1	31	31	20	<1
DPFM(8)	87	321	426	209	162	<1	89	209	162	2
DPFM(11)	1047	5633	2433	1211	1012	<1	313	1211	1012	50
DPH(5)	48	67	5908	2949	1389	<1	70	81	26	4
DPH(6)	57	92	31206	15597	8154	1	84	109	40	13
DPH(7)	66	121	152618	76302	43451	19	98	141	57	48
ELEV(1)	63	99	296	157	59	<1	67	70	24	<1
ELEV(2)	146	299	1562	827	331	<1	145	217	80	12
ELEV(3)	327	783	7398	3895	1629	<1	—	—	—	>15hrs
FURN(1)	27	37	646	394	235	<1	55	62	27	1
FURN(2)	40	65	8123	4980	3331	<1	89	177	74	38
FURN(3)	53	99	103186	63172	43454	8	123	400	166	495
GASNQ(3)	143	223	2601	1301	437	<1	159	245	94	6
GASNQ(4)	258	465	19192	9597	3580	1	280	509	208	35
GASNQ(5)	428	841	127191	63597	24231	23	455	929	399	184
GASQ(1)	28	21	43	21	4	<1	33	19	4	<1
GASQ(2)	78	97	346	173	54	<1	87	85	22	1
GASQ(3)	284	475	2593	1297	490	<1	301	409	108	59
GASQ(4)	1428	2705	26880	13441	5636	1	—	—	—	>15hrs
HART(50)	252	152	354	202	1	<1	303	202	1	8
HART(75)	377	227	529	302	1	<1	453	302	1	24
HART(100)	502	302	704	402	1	<1	603	402	1	57
KEY(2)	94	92	1007796	503896	18620	43559	148	418	61	392
KEY(3)	129	133	—	—	—	>15hrs	—	—	—	>15hrs
MMGT(1)	50	58	118	58	20	<1	61	58	20	<1
MMGT(2)	86	114	2324	1178	493	<1	106	266	105	15
MMGT(3)	122	172	76720	39084	17669	6	151	628	255	371
MMGT(4)	158	232	—	—	—	>15hrs	196	1144	470	15046
OVER(2)	33	32	83	41	10	<1	42	29	7	<1
OVER(3)	52	53	581	296	81	<1	70	65	15	1
OVER(4)	71	74	3040	1556	495	<1	98	109	23	4
OVER(5)	90	95	18648	9551	3485	<1	126	161	31	857
RING(5)	65	55	339	167	37	<1	93	70	15	1
RING(7)	91	77	813	403	79	<1	135	108	23	5
RING(9)	117	99	1599	795	137	<1	177	146	31	14
RW(6)	33	85	806	397	327	<1	51	85	57	1
RW(9)	48	181	9272	4627	4106	<1	75	181	126	88
RW(12)	63	313	98378	49177	45069	1	99	313	222	17048
SENT(50)	179	80	467	248	39	<1	192	101	14	1
SENT(75)	254	105	542	273	39	<1	267	126	14	2
SENT(100)	329	130	617	298	39	<1	342	151	14	4

(at the price of generating larger prefixes). The issue of using local vs. all configurations as cut-off correspondents in the context of unfoldings was investigated in [10]; the main conclusion was that by using all rather than local configurations, smaller prefixes can be produced, but at the expense of significant (often by an order of magnitude) increase in runtime; hence, using all configurations does not pay off unless there are overriding reasons to have as small a prefix as possible. In our experiments, though the MPs were built using all configurations as potential cut-off correspondents, the unfolding prefixes were built using only local ones, which results in inconsistency of the comparison, giving advantage to the unfolder. Unfortunately, no unfolder we are aware of can use all configurations as cut-off correspondents ([10] post-processed the standard prefixes).

Table 1 shows the experimental results. The popular set of benchmarks collected by J.C. Corbett [4] has been attempted, and a PC with an Intel i7 2.8GHz CPU, 4GB RAM and 64-bit Windows 7 was used. Both the unfolder and the unraveller were compiled as 32-bit applications, and no parallelism was used, i.e. only one CPU core was utilised. The unfolding prefixes in our experiments were built using the Punf unfolder [22], and the MiniSat [19] SAT solver was employed by the proposed unravelling algorithm (the 2QBF solver was implemented using a pair of SAT solvers). The meaning of the columns is as follows (from left to right): the name of the problem; the number of places and transitions in the original PN; the number of conditions, events (including cut-offs) and cut-off events in the unfolding prefix, together with the time to construct it; the number of mp-conditions, mp-events (including cut-offs) and cut-off mp-events in the corresponding MP, together with the time to construct it.

As one can see, our implementation of the proposed unravelling algorithm still loses to the unfolder, though there are some successes like Byz, Key(2) and Mmgt(4). These results should be considered in the light of the fact that our implementation is just a prototype, and it is compared with a fully-fledged tool implementing some sophisticated unfolding techniques that took over a decade to develop. Hence, they should be viewed as a proof-of-concept, and as such they do look very promising: we are confident that our implementation can be considerably improved (some possible optimisations that are likely to significantly increase the performance are described in Sect. 5).

5 Conclusions and Future Work

In this paper we have proposed an algorithm for constructing complete MPs of safe PNs. This algorithm is not based on building an intermediate complete unfolding prefix, which is usually exponentially larger than the final result, avoiding thus the main disadvantage of the previous approach [13]. In particular, the challenging problem of identifying cut-offs in MPs has been solved.

A prototype implementation of the algorithm has been evaluated on a number of benchmarks. Though overall it performs worse than unfolding algorithms, it was very successful on some benchmarks. We consider the proposed algorithm as a proof-of-concept, and believe it can be significantly improved. In particular, the following obvious steps can be done:

- The current implementation often re-generates similar SAT instances from scratch, spending considerable time on this. In fact, according to the profiling data, it spends more time generating SAT instances than solving them; we believe this can be significantly improved by making a better use of incremental SAT, which would also yield additional speedup in satisfiability checking.
- Our implementation of the 2QBF solver was very basic. Using a better solver (e.g. implementing non-critical signal reasoning, which can increase the performance by an order of magnitude according to experiments in [23]) is likely to improve the runtime dramatically.
- The SAT encoding of the ERV adequate order [9] would be too large and inefficient, and this was the reason we used the size-lexicographical adequate order in our implementation, which is not total. We believe that another total adequate order, e.g. one similar to the $\prec_{sl}^{d}$ in [8], can be encoded much better, yielding significant speedups due to smaller MPs.
- The proposed algorithm can be easily parallelised, as typically there are several SAT or 2QBF instances that can be solved concurrently. This opportunity is very useful, as contemporary PCs are multi-core. Moreover, the computation can be easily distributed over a network of PCs.

The above improvements seem fairly obvious and just require implementation work. There are also some research areas, progress in which would improve the proposed algorithm:

- The implemented encoding of the $\mathcal{ACYCLIC}$ constraint [13] seems very inefficient; improving it will have a direct effect on both the unravelling algorithm and the subsequent model checking.
- The proposed algorithm works by considering configurations of larger and larger sizes (cf. the *conf_sz* variable). Hence, each iteration of the main loop repeats much of the work done in the previous ones, and so this might be not the best strategy — perhaps this repeated work can be eliminated. Alternatively, the experience from bounded model checking might be useful to alleviate this problem.
- The last few iterations of the algorithm are usually free-running, i.e. no mp-events are added to the MP being built, as the termination criterion is not sharp. These iterations are quite expensive (as the MP has already reached its full size), and so much time is wasted. Hence, developing a sharper termination criterion is likely to significantly improve the runtime.

On a more general note, we believe that the 2QBF based strategy used by the algorithm for identifying cut-offs is fairly general and can be employed elsewhere. Indeed, for an (mp-)event to have multiple local configurations is quite common, e.g. this is the case for unfoldings of PNs with read arcs [1,26], symbolic unfoldings [2] and unfoldings of time nets [3].

Finally, the natural question about generalisation of the proposed algorithm from safe to bounded PNs is more complicated then one might think, for the following reasons. First of all, the mp-configurations become multisets, and so

one would have to use Integer Linear Programming (ILP) instead of SAT, and formulate (and develop an efficient algorithm for) an integer analog of 2QBF. Moreover, the relationship between the configurations of the unfolding and mp-configurations of the unravelling is somewhat more complicated, in particular the isomorphism is lost. Furthermore, no easily checkable characterisation of mp-configurations has been developed yet, which is probably the most serious obstacle (see [13] where this question is expounded, along with some ideas about developing such a characterisation).

Acknowledgements. The authors would like to thank Javier Esparza and Keijo Heljanko for helpful discussions, and to anonymous reviewers for several useful suggestions. This research was supported by the EPSRC grant EP/G037809/1 (VERDAD).

References

1. Baldan, P., Corradini, A., König, B., Schwoon, S.: McMillan's complete prefix for contextual nets. In: Jensen, K., van der Aalst, W.M.P., Billington, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, pp. 199–220. Springer, Heidelberg (2008)
2. Chatain, T., Jard, C.: Symbolic diagnosis of partially observable concurrent systems. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 326–342. Springer, Heidelberg (2004)
3. Chatain, T., Jard, C.: Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 125–145. Springer, Heidelberg (2006)
4. Corbett, J.C.: Evaluating deadlock detection methods for concurrent software. IEEE Trans. Softw. Eng. 22, 161–180 (1996)
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 333–336. Springer, Heidelberg (2004)
6. Engelfriet, J.: Branching processes of Petri nets. Acta Inf. 28, 575–591 (1991)
7. 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)
8. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2008)
9. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. FMSD 20(3), 285–310 (2002)
10. Heljanko, K.: Minimizing finite complete prefixes. In: Proc. CS&P 1999, pp. 83–95 (1999)
11. Heljanko, K.: Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. Fund. Inf. 37, 247–268 (1999)
12. Khomenko, V.: Model Checking Based on Prefixes of Petri Net Unfoldings. Ph.D. thesis, School of Comp. Sci., Newcastle Univ. (2003)
13. 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)

14. Khomenko, V., Koutny, M., Vogler, V.: Canonical prefixes of Petri net unfoldings. Acta Inf. 40(2), 95–118 (2003)
15. Khomenko, V., Mokhov, A.: An algorithm for direct construction of complete merged processes. Tech. Rep. CS-TR-1231, School of Comp. Sci., Newcastle Univ. (2011), http://www.cs.ncl.ac.uk/publications/trs/papers/1231.pdf
16. McMillan, K.: Symbolic Model Checking: an Approach to the State Explosion Problem. Ph.D. thesis, School of Comp. Sci., Carnegie Mellon Univ. (1992)
17. McMillan, K.L.: 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)
18. Melzer, S., Römer, S.: Deadlock checking using net unfoldings. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 352–363. Springer, Heidelberg (1997)
19. MINISAT tool home page, http://minisat.se
20. Moskewicz, S., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. DAC 2001, pp. 530–535. ASME Technical Publishing (2001)
21. Murata, T.: Petri nets: Properties, analysis and applications. Proc. of the IEEE 77(4), 541–580 (1989)
22. PUNF home page, http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf
23. Ranjan, D., Tang, D., Malik, S.: A comparative study of 2QBF algorithms. In: SAT 2004, ACM, New York (2004)
24. Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comp. Sci. 3(1), 1–22 (1976)
25. Valmari, A.: The State Explosion Problem. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998)
26. Vogler, W., Semenov, A., Yakovlev, A.: Unfolding and finite prefix for nets with read arcs. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 501–516. Springer, Heidelberg (1998)
27. Zhang, L., Malik, S.: The quest for efficient Boolean satisfiability solvers. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 582–595. Springer, Heidelberg (2002)

How Much Is Worth to Remember? A Taxonomy Based on Petri Nets Unfoldings*

G. Michele Pinna

Dipartimento di Matematica e Informatica
Università degli Studi di Cagliari, Cagliari, Italy
gmpinna@unica.it

Abstract. The notion of unfolding plays a major role in the so called *non sequential* semantics of Petri nets, as well as in model checking of concurrent and distributed systems or in control theory. In literature various approaches to this notion have been proposed, where dependencies among events are represented either taking into account the whole history of the event (the so called *individual token philosophy*) or considering the whole history irrelevant (the so called *collective token philosophy*). In this paper we propose two unfoldings where the history is partially kept. These notions are based on *unravelling* a net rather than *unfolding* it. We compare them with the classical ones and we put all of them together in a coherent framework.

1 Introduction

Petri nets are since almost 40 years one among the most widely used models for concurrent and distributed computations.

This kind of computations can be described at different granularities, remembering to some extent *how* events happen, which in the case of Petri nets usually means either considering the tokens consumed and produced by happening of transitions as *entities* coding the history (namely the occurrences of transitions and the ordering of these occurrences, that have *contributed* to produce the specific token) or simply as *anonymous* ones (the history is forgotten). Henceforth to describe (and to reason on) computations in this setting two main approaches are usually taken: one asserts that each happening of a transition depends on how tokens are consumed and produced by other transitions, and a second declares that the happening of a transitions depends only on the fulfillment of certain conditions, namely the presence of tokens in the preset of the transition, without any concern on how these conditions were produced. In other words, one approach considers the history relevant whereas the other considers the history irrelevant.

* Work partially supported by the *CGR Coarse Grain Recommendation*-project, PIA "Industria, Artigianato e Servizi", Regione Autonoma della Sardegna, and TESLA, L.R. 7/2007, Regione Autonoma della Sardegna.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 109–128, 2011.

To make more precise this concept, consider the net in Fig. 1. In the net N the transition a depends on the happening of b or c, and if the history would be considered relevant, the information on which of b or c is actually executed must be kept, whereas if the history here does not play a major role then the information on *which of the two transitions* was executed is negligible. The history respecting approach is mainly represented by the classical notion of unfolding (which will be called I-unfolding) as introduced by Winskel ([1])[1]. In this case the transition a depending on b is *different* from the transition a depending on c. This view is reflected naturally in the again classical notion of *prime event structure* ([1]). Concerning the other approach, van Glabbeek and Plotkin in [5] introduced the notion of 1-unfolding (which we call C-unfolding) to model the irrelevance of the history and contextually introduced a notion of event structures, called *configuration structures*, where this principle is reflected.

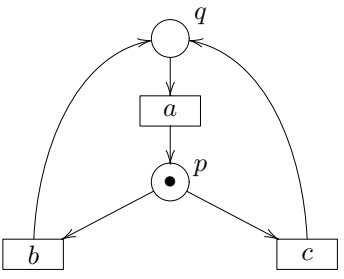


Fig. 1. The net N

In this paper the focus is on the possibility that tokens code just a *part* of the history, hence on notions of unfolding where only parts of the history are kept.

Other authors have addressed this issue, mainly inspired by Petri Nets but focussing on other models: Gunawardena in [6] introduced a notion, called *Muller unfolding* (for the net N this notion is based on the intuition that the $i-th$ happening of a depends on the happening of $i-1$ occurrences of b or c)[2], and introduced a suitable event based semantics, expressing a principle of *history irrelevance*. In fact already in [7] and [8] Gunawardena advocates the principle of irrelevance of the whole history by introducing *causal automata*, where the so called *or-causality* can be easily modeled, meaning that an event may have several *alternative* causes. In [9] a notion of event structure is introduced, where the dependencies among events are modeled in a local fashion, implementing again a limited principle of history irrelevance, whereas in [10] a notion of scenario is introduced to adjust some weaknesses of classical partial order semantics for Petri nets, where again the history is not posed as the central notion.

Turning to behaviours described in term of suitable nets, Khomenko, Kondratyev, Koutny and Vogler in [11] have developed a way to obtain a more compact representation of the net behaviours, basically identifying all the conditions representing the *same* token and identifying as well the transitions that consume and produce the same conditions (i.e., tokens). Their target was mainly to achieve representations which are more compact, and forgetting part of the

[1] Also other authors have investigated on this notion, e.g. Engelfriet [2] for the unfolding of safe nets, or Baldan, Corradini and Montanari in [3] for the unfolding of net with read arcs, or Baldan, Busi, Corradini and the author in [4] for unfoldings of safe nets with inhibitor and read arcs.

[2] For Gunawardena the word *unfolding* does not refer to a net.

history leads to this kind of results. Fabre in [12] introduces the notion of *Trellis processes*, which is basically an unfolding where conflicting occurrences of a transition are identified, where conflicting means belonging to different computations. Trellis processes are defined for a suitable class of safe nets, namely nets that can be obtained by the union of state-machines, i.e., condition/events nets. Though some similarities with our approach exists, in Trellis a transition codes histories which can be considered as adhering to the individual token philosophy.

In [13] van Glabbeek analyzes the computational interpretations of various firing rules for Petri Nets, distinguishing not only between the individual token approach (the one where history is relevant) and the collective token one (where history is not relevant), but also the fact that a transition may be concurrent with itself. Here we rule out auto-concurrency and then we focus on *safe* nets, i.e., nets where each place in each reachable marking contains at most one token.

The notions of unfoldings introduced in literature are based on suitable nets enjoying *structural* and *behavioural* restrictions: the individual unfoldings on *causal* net, i.e., acyclic nets which can be equipped with an irreflexive conflict relation, the collective unfolding on the notion of *occurrence* net (which can be cyclic), where it is guaranteed that each transition fires just once. The notions of unfolding we will introduce are based on *unravel* nets, namely occurrence nets which are acyclic only when considering suitable subsets of events (representing a computation). Concerning the two new notions of unfoldings, one is based on the observation that the i-th occurrences of a token in a place can be equated (thus in this unfolding, which we will call R_i-unfolding, relative to the net in Fig. 1 the token produced in the place q by either c or b is represented by a unique condition); whereas the other considers also that the all the events happening at the same *depth* (a distance from the initial marking) and having the same *surrounding* can be identified (thus in this unfolding, which we will call R_c-unfolding, relative to the net in Fig. 1 the two occurrence of the transition a, one depending on b and the other on c are identified).

These notions could be obtained by first constructing the individual unfolding, and then identifying suitable places and transitions. We do take another way, as we define these unfolding from scratch, without constructing first the individual unfolding, which may be a complex operation. When identifying the events of both unfoldings we face the problem of determining the *amount* of history that should be recorded: in the R_i-unfolding an event basically codes the count of the transitions in its neighborhood, whereas in the R_c-unfolding it is enough to count just how many times the transition corresponding to the event has been executed.

Beside comparing the two notion of unfoldings introduced in literature, similarly to what van Glabbeek has done in [13] with respect to firing sequences, we discuss also the relative expressiveness of the two new notions. As evidence of the expected relative expressiveness, we will show that what we will call the *individual unfolding*, i.e. the unfolding for the individual token philosophy can be folded onto the other notions of unfoldings, leading to the *collective unfolding* (i.e. the unfolding for the collective token philosophy). The results can be

summarized as follow: the I-unfolding can be folded onto the R_i-one, which in turn is folded onto the R_c-unfolding and finally onto the C-unfolding. This amount to saying that histories can be progressively *faded*, i.e., forgotten. In [14] we have shown also that the collective unfolding can be unfolded onto the individual one. And this account to say that the history can be always recovered.

The paper is organized as follows: in the next section we recall the notions we will use in the rest of the paper, and introduce new ones, like the one of unravel net. In section 3 we introduce the notions of unfolding which we will compare in section 4. We sum up our finding in the last section.

2 Nets

Notations: With $\mathbb{N}$ we denote the set of natural numbers and with $\mathbb{N}^+$ the set of natural numbers without zero, i.e., $\mathbb{N} \setminus \{0\}$. Let X be a set and $n \in \mathbb{N}^+$, with X^n we denote the set of n-tuple of elements in X. Given a tuple $\alpha \in X^n$, with $\alpha(i)$ we indicate the i-th entry of the tuple. Let A be a set, a *multiset* of A is a function $m : A \to \mathbb{N}$. The set of multisets of A is denoted by μA. The usual operations and relations on multisets, like multiset union $+$ or multiset difference $-$, are used. We write $m \leq m'$ if $m(a) \leq m'(a)$ for all $a \in A$. If $m \in \mu A$, we denote by $[\![m]\!]$ the multiset defined as $[\![m]\!](a) = 1$ if $m(a) > 0$ and $[\![m]\!](a) = 0$ otherwise; sometimes $[\![m]\!]$ will be confused with the corresponding subset $\{a \in A \mid [\![m]\!](a) = 1\}$ of A. A *multirelation* f from A to B (often indicated as $f : A \to B$) is a multiset of $A \times B$. We will limit our attention to finitary multirelations, namely multirelations f such that the set $\{b \in B \mid f(a, b) > 0\}$ is finite. Multirelation f induces in an obvious way a function $\mu f : \mu A \to \mu B$, defined as $\mu f(\sum_{a \in A} n_a \cdot a) = \sum_{b \in B} \sum_{a \in A} (n_a \cdot f(a, b)) \cdot b$ (possibly partial, since infinite coefficients are disallowed). If f satisfies $f(a, b) \leq 1$ for all $a \in A$ and $b \in B$, i.e. $f = [\![f]\!]$, then we sometimes confuse it with the corresponding set-relation and write $f(a, b)$ for $f(a, b) = 1$.

Given an alphabet Σ, with Σ^* we denote as usual the set of words on Σ, and with $\preceq$ we denote the lexicographic order on it. The length of a word is defined as usual and it is denoted with $[\cdot]$. The number of occurrences of a symbol $a \in \Sigma$ in a word w is standardly defined as $[w]_a = 0$ if $w = \epsilon$, $[w]_a = [x]_a + 1$ if $w = ax$, $[w]_a = [x]_a$ otherwise. Given a word w and a subset A of the alphabet, $\|w\|_A$ is a word obtained deleting all occurrences of symbols not belonging to A, i.e. $\|w\|_A = a\|x\|_A$ if $a \in A$ and $w = ax$, and $\|w\|_A = \|x\|_A$ otherwise. Finally, given a word $w \in \Sigma^+$, with $last(w)$ we denote the last symbol appearing in w.

Given a partial order $(D, \sqsubseteq)$, with $\lfloor d \rfloor$ we denote the set $\{d' \in D \mid d' \sqsubseteq d\}$.

Nets and morphisms: We first review the notions of Petri net and of the token game. Then we recall the notion of morphism. Nets and morphism are a category, but our focus is not on the categorical approach to nets, but rather on the various relations among the notions of unfolding that are definable on nets. In this perspective morphisms are quite handy in highlighting how nets (unfoldings) are related.

Definition 1. *A* Petri net *is a 4-tuple* $N = \langle S, T, F, m\rangle$*, where*

- S *is a set of* places *and* T *is a set of* transitions *(with* $S \cap T = \emptyset$*),*
- $F = \langle F_{pre}, F_{post}\rangle$ *is a pair of multirelations from* T *to* S*, and*
- $m \in \mu S$ *is called the* initial marking*.*

We require that for each $t \in T$, $F_{pre}(t, s) > 0$ for some place $s \in S$. Subscripts on the net name carry over the names of the net components. As usual, given a finite multiset of transitions $A \in \mu T$ we write ${}^\bullet A$ for its *pre-set* $\mu F_{pre}(A)$ and $A^\bullet$ for its *post-set* $\mu F_{post}(A)$. The same notation is used to denote the functions from S to $\mathbf{2}^T$ defined as ${}^\bullet s = \{t \in T \mid F_{post}(t, s) > 0\}$ and $s^\bullet = \{t \in T \mid F_{pre}(t, s) > 0\}$, for $s \in S$.

Let N be a net. A finite multiset of transitions A is enabled at a marking m, if m contains the pre-set of A. Formally, a finite multiset $A \in \mu T$ is *enabled* at m if ${}^\bullet A \leq m$. In this case, to indicate that the execution of A in m produces the new marking $m' = m - {}^\bullet A + A^\bullet$ we write $m\,[A\rangle\, m'$. Steps and firing sequences, as well as reachable markings, are defined in the usual way. The set of reachable markings of a net N is denoted with $\mathcal{M}_N$. Each reachable marking can be obviously reached with a firing sequence where just a transition is executed at each step. Thus also a trace can be associated to it, which is the word on T^* obtained by the firing sequence considering just the transitions and forgetting about the markings. The traces of a net N are denoted with $Traces(N)$.

We recall now the notion of morphism between nets, which we will use to construct an unfolding (and also to relate the various notions of unfolding).

Definition 2. *Let* $N_0 = \langle S_0, T_0, F_0, m_0\rangle$ *and* $N_1 = \langle S_1, T_1, F_1, m_1\rangle$ *be nets. A* morphism $h : N_0 \to N_1$ *is a pair* $h = \langle \eta, \beta\rangle$*, where* $\eta : T_0 \to T_1$ *is a partial function and* $\beta : S_0 \to S_1$ *is a multirelation such that*

- $\mu\beta(m_0) = m_1$*,*
- *for each* $t \in T$*,* $\mu\beta({}^\bullet t) = {}^\bullet\eta(t)$*, and* $\mu\beta(t^\bullet) = \eta(t)^\bullet$*.*

The conditions of the above definition are the defining conditions of Winskel's morphisms on ordinary nets ([1]). A morphism $h : N_0 \to N_1$ in this setting is a *simulation*, in the sense that each reachable marking in N_0 is also a reachable marking in N_1.

Proposition 1. *Let* N_0 *and* N_1 *be nets, and let* $h = \langle \eta, \beta\rangle : N_0 \to N_1$ *be a net morphism. For each* $m, m' \in \mathcal{M}_{N_0}$ *and* $A \in \mu T$*, if*

$$m\,[A\rangle\, m' \text{ then } \mu\beta(m)\,[\mu\eta(A)\rangle\,\mu\beta(m').$$

Therefore net morphisms preserve reachable markings, i.e. if m *is a reachable marking in* N_0 *then* $\mu\beta(m)$ *is reachable in* N_1*.*

Safe nets: In this paper we consider only safe nets:

Definition 3. *A net* $N = \langle S, T, F, m\rangle$ *is said* safe *in the case that* F_{pre} *and* F_{post} *are such that* $F_{pre} = [\![F_{pre}]\!]$*,* $F_{post} = [\![F_{post}]\!]$ *and each marking* $m \in \mathcal{M}_N$ *is such that* $m = [\![m]\!]$*.*

Furthermore we require that the safe nets are such that $\forall s \in S, \forall t \in T$ it holds that $F_{pre}(t, s) \neq F_{post}(t, s)$.

Subnet: A subnet of a net is a net obtained restricting places and transitions, and correspondingly also the multirelation F and the initial marking.

Definition 4. *Let $N = \langle S, T, F, m\rangle$ be a Petri net and let $T' \subseteq T$. Then the subnet generated by T' is the net $N|_{T'} = \langle S', T', F', m', \rangle$, where*

- $S' = \{s \in S \mid F_{pre}(t,s) > 0$ *or* $F_{post}(t,s) > 0$ *for* $t \in T'\} \cup \{s \in S \mid m(s) > 0\}$,
- $F' = \langle F'_{pre}, F'_{post}\rangle$ *is the pair of multirelations from T' to S' obtained restricting $\langle F_{pre}, F_{post}\rangle$ to S' and T',*
- m' *is the multiset on S' obtained by m restricting to places in S'.*

It is trivial to observe that, given a net $N = \langle S, T, F, m\rangle$ and a subnet $N|_{T'}$, with $T' \subseteq T$, then the pair $\eta(t) = t$ and $\beta : S \to S'$ is $\beta(s,s) = 1$ if $s \in S'$ and 0 otherwise is a well defined morphism from N to $N|_{T'}$.

Occurrence Nets: The notion of occurrence net we introduce here is the one called 1-occurrence net and proposed by van Glabbeek and Plotkin in [5]. First we need to introduce the notion of *state*.

Definition 5. *Let $N = \langle S, T, F, m\rangle$ be a Petri net, the* configuration *is any finite multiset X of transitions with the property that the function $m_X : S \to \mathbb{Z}$ given by $m_X(s) = m(s) + \sum_{t\in T} X(t) \cdot (F_{post}(t,s) - F_{pre}(t,s))$, for all $s \in S$, is a marking of the net.*

Given two configurations X, Y, we stipulate that $X \xrightarrow{A} Y$ iff $m_X\,[A\rangle\, m_Y$. A configuration X is reachable iff $X = \oplus_{i=1}^{n} X_i$ is such that $\oplus_{j=1}^{k-1} X_j \xrightarrow{X_k} \oplus_{j=1}^{k} X_j$ for all $1 \leq k \leq n$. If X is a reachable configuration, we call it a state. *With $\mathcal{X}(N)$ we denote the states of a net.*

A state contains (in no order) all the occurrence of the transitions that have been fired to reach a marking. Observe that a trace of a net is a suitable linearization of the elements of a state X. On the notion of state the notion of occurrence net is based:

Definition 6. *An* occurrence net *$O = \langle S, T, F, m\rangle$ is a Petri net where each state is a set, i.e. $\forall X \in \mathcal{X}(N)$ it holds that $X = [\![X]\!]$.*

The intuition behind this notion is the following: regardless how tokens are produced or consumed, an occurrence net *guarantees* that each transition can *occur* only once (hence the reason for calling them occurrence nets). As suggested in the introduction, the history of a token (how it is produced) is completely forgotten. Observe that, differently from the notion of causal net we will introduce later, an occurrence net is not required in general to be a safe net. Another relevant characteristic is that it is not possible (or at least easy) to obtain directly relationships among occurrences of transitions, as it will be with causal nets.

Causal Nets: The notion of causal net we use here is the classical one, though it is often called occurrence net. The different name is due to the other notion of occurrence net we have introduced above.

Causal nets are structurally safe nets, hence the multirelations F_{pre} and F_{post} can be considered as a *flow* relation $F \subseteq (S \times T) \cup (T \times S)$ by stating $s\ F\ t$ iff $F_{pre}(t,s)$ and $t\ F\ s$ iff $F_{post}(t,s)$. Hence we can use the usual notation for the transitive (and reflexive) closure of this relation. For denoting places and transitions we use B and E (see [15] and [1,16]). A causal net is essentially an acyclic net equipped with a conflict relation (which is deduced using the transitive closure of F).

Definition 7. *A* causal net $C = \langle B, E, F, m\rangle$ *is a safe net satisfying the following restictions:*

- $\forall b \in m,\ {}^\bullet b = \emptyset$,
- $\forall b \in B.\ \exists b' \in m$ *such that* $b' F^* b$,
- $\forall b \in B.\ |{}^\bullet b| \leq 1$,
- F^+ *is irreflexive and, for all* $e \in E$, *the set* $\{e' \mid e'\ F^* e\}$ *is finite, and*
- $\#$ *is irreflexive, where* $e \#_i e'$ *iff* $e, e' \in E$, $e \neq e'$ *and* ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$, *and* $x \# x'$ *iff* $\exists y, y' \in B \cup E$ *such that* $y \#_i y'$ *and* $y F^* x$ *and* $y' F^* x'$.

The intuition behind this notion is the following: each place b represents the occurrence of a token, which is produced by the *unique* transition in ${}^\bullet b$, unless b belongs to the initial marking, and it is used by only one transition (hence if $e, e' \in b^\bullet$, then $e \# e'$). On causal nets it is easy to define a relation expressing *concurrency*: two elements of the causal net are concurrent if they are neither causally dependent nor in conflict. Formally $x\ co\ y$ iff $\neg(x \# y$ or $x F^+ y$ or $y F^+ x)$. This relation can be extended to sets of conditions: let $A \subseteq B$, then $\mathbf{co}(A)$ iff $\forall b, b' \in A.\ b\ co\ b'$ and $\{e \in E \mid \exists b \in A.\ e F^* b\}$ is finite.

The following observation essentially says that each transition (event) in a causal net is executed only once:

Proposition 2. *let* $C = \langle B, E, F, m\rangle$ *be a causal net, then* C *is also an occurrence net.*

Unravel Nets: Causal nets structurally capture dependencies (and conflict) whereas occurrence nets structurally capture the unique occurrence property of each transition. We introduce now a notion of net which will turn to be, so to say, in between occurrence nets and causal nets. With respect to the notion of occurrence net we want still to structurally assure that each transition happens just once, whereas with respect to causal net we want still to be able to retrieve dependencies among the firings of transitions.

Definition 8. *An* unravel net $R = (\langle S, T, F, m\rangle, \mathcal{P})$ *is a pair where:*

- $\langle S, T, F, m\rangle$ *is an occurrence net,*
- *for each state* $X \in \mathcal{X}(\langle S, T, F, m\rangle)$ *such that* $\mathcal{P}(X)$, *the restriction of* $\langle S, T, F, m\rangle$ *to the transitions in* $[\![X]\!]$, *i.e.,* $\langle S, T, F, m\rangle|_{[\![X]\!]}$, *is a causal net.*

Given and unravel net $R = (\langle S, T, F, m\rangle, \mathcal{P})$, with abuse of notation we will often use R for $\langle S, T, F, m\rangle$.

As discussed previously, the intuition behind an unravel net is the similar of the *merged* processes in ([11]) where each place of the net represents a token (as in causal ones) but it can be produced in various ways (the possible histories of the token) and these histories can be recovered once one considers a state of the net. Thus on the one hand, the idea is that the i-th presence of a token in a place is *signaled* just once; and on the other hand the states of the nets satisfying a property (which we left here unspecified) leads to a causal net. Observe that, given an unravel net R and a state X satisfying the required property, $R|_{[\![X]\!]}$ is a causal net where the conflict relation is empty.

This notion covers the one of causal net, as the following proposition shows.

Proposition 3. *Let $C = \langle B, E, F, m\rangle$ be a casual net. Then $R = (C, \mathcal{P}_{\leq})$ is an unravel net, where $\mathcal{P}_{\leq}(X)$ holds iff for all $t \in [\![X]\!]$. $\lfloor x \rfloor \subseteq [\![X]\!]$, with respect to (E, F^*).*

It is worth to observe that the choice of the property turning causal nets into unravel nets is simple. Furthermore other properties on transitions can be used in this case: for instance, the one holding for each subset of transitions, or the one requiring that $[\![X]\!]$ is conflict free, i.e., for all $t, t' \in [\![X]\!]$ it holds that $(t, t') \notin \#$, where $\#$ is the conflict relation induced by the causal net.

3 Unfoldings

In this section we introduce various notions of unfoldings of a net, related to occurrence, unravel and causal nets. Two of these constructions are *classic* and are closely related to the individual and collective token philosophy, the others are new. In the following we will use the same notation both for the multirelations and for the relations between transitions (events) and places (conditions). This can be done safely as on causal nets these coincide.

Individual unfolding: In the case of the individual token philosophy the unfolding of a net N is a causal net. It can be defined either as top of a chain of causal nets or as the unique causal net satisfying certain conditions.

Definition 9. *Let $N = \langle S, T, F, m\rangle$ be a safe net. The I-*unfolding *$\mathcal{U}_I(N) = \langle B^I, E^I, F^I, m^I\rangle$ is the net defined as follows:*

$$B^I = \{(m, s) \mid s \in S \text{ and } m(s) > 0\} \bigcup \{\{(\{e\}, *) \mid e \in E^I\} \bigcup \{(\{e\}, s) \mid e \in E^I \text{ and } s \in S \text{ and } F^I_{pre}(\eta^I(e), s) > 0\}$$

$$E^I = \{(X, t) \mid X \subseteq B^I \text{ and } \mathbf{co}(X) \text{ and } {}^\bullet t = \mu\beta^I(X)\}$$

$$F^I = \begin{cases} F^I_{pre}((X,t), b) \text{ iff } b \in X \text{ or } b = (\{(X,t)\}, *) \\ F^I_{post}((X,t), b) \text{ iff } \exists s \in S, i \in \mathbb{N}^+.\ b = ((X,t), s) \end{cases}$$

$$m^I = \{(m, s) \mid (m, s) \in B^I\} \bigcup \{\{(\{e\}, *) \mid e \in E^I\}$$

where **co** *is the concurrency relation obtained by* F^I *on* B *and* E, $\eta : E^I \to T$ *is defined as* $\eta^I(X,t) = t$ *and* $\beta : B^I \to S$ *is defined as* $\beta^I(X,s) = s$.

In this construction each token carries the whole history, and the events as well. As we said before, there are other alternative characterizations of the I-unfolding of a net.

The construction we have introduced has a little difference with respect to the ones introduced in the literature (e.g. [1] or [2]), as we introduce a condition for each event (transition), namely $(\{e\}, *)$, which enforces the unique occurrence property of each event (transition). This cause no harm (these conditions are superfluous, in a net theoretical sense, for causal nets) but they are useful in relating the unfolding constructions.

Proposition 4. *Let* N *be a safe net and* $\mathcal{U}_I(N)$ *its* I-unfolding. *Then* $\mathcal{U}_I(N)$ *is a causal net and* $\langle \eta^I, \beta^I \rangle : \mathcal{U}_I(N) \to N$ *is a well defined net morphism.*

A consequence of this construction is that each reachable marking of the unfolding is a reachable marking of the net (this is due to the folding morphism) but also the converse holds, i.e. to each reachable marking of the net N a reachable marking in the unfolding corresponds.

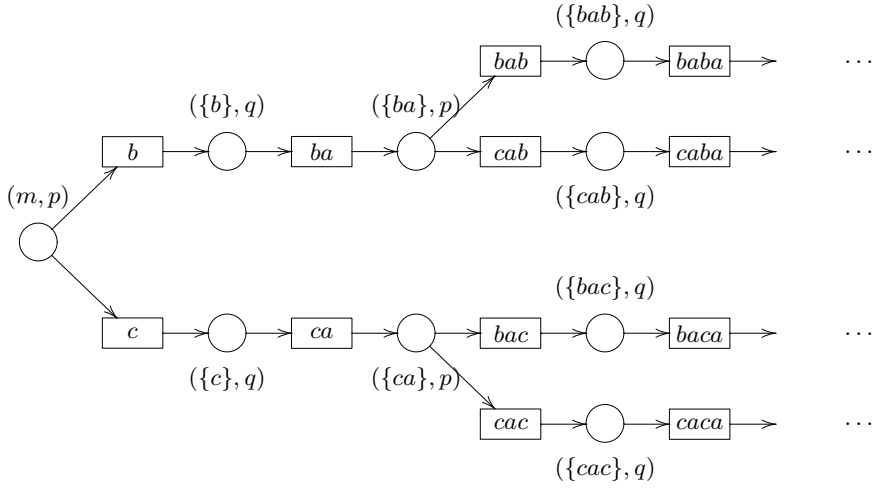


Fig. 2. Part of the I-unfolding of N (places $(t, *)$ are omitted)

R_i-unfolding: We introduce now the first of the two new notions of unfolding of a net, which will turn out to be an unravel net (and we will denote it with $\mathcal{U}_{R_i}$). In order to construct this unfolding, we need some auxiliary notations. Let $N = \langle S, T, F, m \rangle$ be a safe Petri net, and let $t \in T$, with $\circlearrowleft(t) = \{t' \mid t' \in {}^\bullet s$ or $t' \in s'^\bullet$ with $s \in {}^\bullet t$ and $s' \in t^\bullet\} \cup \{t\}$ we denote the *neighborhood* of t, namely the transitions *following* and *preceding* t, including t.

The places of this unfolding are easy to identify: consider a safe net $N = (S, T, F, m)$, then the places are $\{(s, i) \mid s \in S \text{ and } i \in \mathbb{N}^+\}$, meaning that the place s got its i-th token.

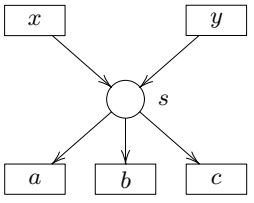


Fig. 3. Part of a safe net

To identify the transitions, let us consider the part of the net depicted in Fig. 3. The i-th firing of the transition a (consuming the j-th token put in s) depends on the number of occurrences of x, y, b and c (and a itself). It clearly depends on x and y as they produce tokens in s, but also on b, c and the previous occurrences of a, as these remove tokens from s. Thus we have to collect in some way this information, which is directly available on the individual unfolding (by looking at the past of the events representing the i-th firing of a).

The i-th firing of the transition a (which may have various histories, that for the time being we keep separate) is represented by the event $\{(\!|w|\!)_\sim a \mid w \in (\circlearrowleft(a))^*\}$ where $\sim$ is an equivalence relation stating that $w \sim w'$ iff $|w| = |w'|$ and for all $t \in T$ $[w]_t = [w']_t$, and $(\!|w|\!)_\sim$ denotes the equivalence class of the word w which is such that

- $[w]_a = i - 1$ (before the i-th happening of a, a itself must have happened $i - 1$-th times),
- $[w]_x + [w]_y$ is equal to j if $m(s) = 0$, or it is equal to $j - 1$ is $m(s) = 1$ (x and y must have happened a number of times (j or $j - 1$) in order to put the token the j-th time in s, and
- $[w]_a + [w]_b + [w]_c = j - 1$ (the token has been removed just $j - 1$ times).

This definition of event takes into account what we can call its *local* history of a, by describing the happening of the transitions in its neighborhood (the order is not relevant and can be retrieved looking among the elements in the equivalence class). We often confuse the equivalence class with one of its elements.

To illustrate how transitions and places are connected, we refer again to the Fig. 3. The transition wa happens whenever the places in its preset are marked, which means that if $s \in {}^\bullet a$ in N, then wa can happens when the place (s, j) got marked, with $j = [w]_{{}^\bullet s \cup s^\bullet} + 1$. Assume that x, y, b and c have happened respectively k, m, n, p times. Then the w is the encoding of this fact (and should have length $k + m + n + p + i - 1$). Thus an arc will connect (s, j) to wa in the unravel net. To characterize which event put the j-th token in s, we have to concentrate our attention on x and y. This token in s is put either by x or y, and precisely by possibly any of the occurrence of x and y, wx or $w'y$. where $j = [wx]_{\{x,y\}}$ and $j = [w'y]_{\{x,y\}}$. Thus an arc should connect these events to (s, j).

What remain to describe is the property $\mathcal{P}$. An unravel net can have cycles (as it should be only *locally* acyclic). Notwithstanding, it is easy to require that the subnet associated to a state X is acyclic, just by requiring that the relation F^* restricted to the elements of X is a partial order. Unfortunately this is not

enough. Consider the state of the net in fig. 4 $\{c, ba, bac\}$. The subnet identified by these three events is acyclic, but it is not what we would like to have, at least at this point, as c corresponds in the original net to c and ba conveys the idea that it represents the happening of a after b has happened (and not c as in this case). Thus we require that also the events have to be related, requiring that the events in X can be ordered in a suitable way with respect to their local histories. Given a trace w and an event $(\!|w'|\!)_{\sim}b$, we can *add* the b at the end of the trace whenever $\|w\|_{\circlearrowleft(b)} \in (\!|w'|\!)_{\sim}$, and we denote it with $w \circledcirc b$. Let Y be a finite set of events equipped with a total ordering $\sqsubseteq$, with $\Downarrow_{\sqsubseteq} Y$ we denote the word $(\Downarrow_{\sqsubseteq} (Y \setminus \{max(Y)\})) \circledcirc last(max(Y))$. Putting all together, the property we are looking for on a state X, is the following:

- X is acyclic with respect to $F^* \cap [\![X]\!] \times [\![X]\!]$, and
- there exists a relation $\lhd$ on $[\![X]\!]$ such that
  - $\lhd^*$ is a total order compatible with F^*, and
  - $\Downarrow_{\lhd^*} [\![X]\!] \in \mathit{Traces}(N)$.

We are now ready to propose the following construction:

Definition 10. *Let $N = \langle S, T, F, m\rangle$ be a safe net. The R_i-unfolding $\mathcal{U}_{R_i}(N) = (\langle S^{R_i}, T^{R_i}, F^{R_i}, m^{R_i}\rangle, \mathcal{P}^{R_i})$ is the net defined as follows:*

$$S^{R_i} = (S \times \mathbb{N}^+) \cup (T^{R_i} \times \{*\})$$

$$T^{R_i} = \bigcup_{t \in T} \{(\!|w|\!)_{\sim}t \mid w \in (\circlearrowleft(t))^*\}$$

$$F^{R_i} = \begin{cases} F^{R_i}_{pre}((\!|w|\!)_{\sim}t, (s,k)) & \textit{iff } F_{pre}(t,s) \\ & \textit{and } k = [\|w\|_{s^\bullet \cup {}^\bullet s}] + m^{R_i}(s,1) \\ F^{R_i}_{pre}((\!|w|\!)_{\sim}t, ((\!|w|\!)_{\sim}t, *)) & \\ F^{R_i}_{post}((\!|w|\!)_{\sim}t), (s,k)) & \textit{iff } F_{post}(t,s) \\ & \textit{and } k = [\|w\| \, {}^\bullet s] + m^{R_i}(s,1) + 1 \end{cases}$$

$$m^{R_i} = \{((s,1)) \mid m(s) > 0\} \bigcup \{(e,*) \mid e \in T^{R_i}\}$$

$\mathcal{P}^{R_i}(X)$ *iff X is a state and there exists a total order $\lhd^*$, compatible with $(F^{R_i})^*$, such that $\Downarrow_{\lhd^*} [\![X]\!] \in \mathit{Traces}(N)$.*

Furthermore $\eta^{R_i} : T^{R_i} \to T$ is defined as $\eta((\!|w|\!)_{\sim}t) = t$ and $\beta^{R_i} : S^{R_i} \to S$ is a multirelation defined as $\beta^{R_i}((s,j)) = s$ iff $s \in S$.

Observe that transitions produced by the *firing of the same transitions* in the I-unfolding of a safe net N are identified in the R_i-unfolding (for instance, the two transitions *baca* and *caba* are now the same in this unfolding), and several conditions of the I-unfolding are mapped to a single one in the R_i-unfolding (i.e., those representing the same occurrence of a token in the place). Opposed to the I-unfolding, where each event encodes the whole history, here it encodes just a part of the whole history. The same happening, in different context, of

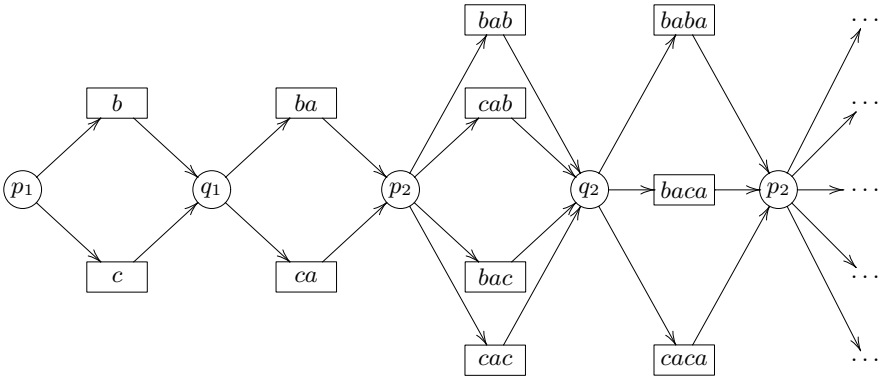


Fig. 4. Part of the R_i-unfolding of N (places $(t, *)$ are omitted)

a transition in the original net is represented by an event for each one (see the events *ca* and *ba* in the unfolding in Fig. 4. This unfolding has the same conditions of the merged process of a safe net defined in [11], as we will see later (some events that could be identified, are not glued in our approach as for the time being a relevant part of the history is preserved).

The following proposition shows that the construction we have proposed is indeed a faithful representation of the behaviour of a net, as to each reachable marking of the net N a state in the unfolding satisfying the suitable property can be found.

Proposition 5. *Let* $N = \langle S, T, F, m\rangle$ *be a safe net and* $\mathcal{U}_{R_i}(N) = (\langle S^{R_i}, T^{R_i}, F^{R_i}, m^{R_i}\rangle, \mathcal{P}^{R_i})$ *its* R_i*-unfolding. Then*

1. $\mathcal{U}_{R_i}(N)$ *is an unravel net,*
2. $\langle \eta^{R_i}, \beta^{R_i}\rangle : \mathcal{U}_{R_i}(N) \to N$ *is a well defined net morphism, and*
3. *for each state* $X \in \mathcal{X}(N)$ *there exists a state* $Y \in \mathcal{X}(\mathcal{U}_{R_i}(N))$ *such that* $\mathcal{P}^{R_i}(Y)$ *and* $\mu\beta^{R_i}(m_Y) = m_X$.

Proof. 1. $\mathcal{U}_{R_i}(N)$ is clearly an occurrence net by construction (each transition t can fire just once, consuming the token in the place $(t, *)$, which is initially marked and has no incoming arcs). It is an unravel net as well. Consider a state $X \in \mathcal{X}(\mathcal{U}_{R_i}(N))$ such that $\mathcal{P}^{R_i}(X)$ and assume that $\mathcal{U}_{R_i}(N)|_{\lfloor X \rfloor}$ is not acyclic. Then there must be a place, say (s, i), which is *marked* twice. Assume that ${}^\bullet s = \{x, y\}$. X must contain two transitions, $(\!|w|\!)_\sim a$ and $(\!|w'|\!)_\sim b$, with $a, b \in \{x, y\}$, putting the token in (s, i). As $\mathcal{P}^{R_i}(X)$ it must be that either $(\!|w|\!)_\sim a \lhd^* (\!|w'|\!)_\sim b$ or $(\!|w'|\!)_\sim b \lhd^* (\!|w|\!)_\sim a$. Assume the former. This means that (s, i) has been produced for the first time by $(\!|w|\!)_\sim a$, but by definition of $\mathcal{U}_{R_i}(N)$ it cannot be produced by $(\!|w'|\!)_\sim b$, as the number or elements in ${}^\bullet s$ in $(\!|w'|\!)_\sim$ is certainly different from those in $(\!|w|\!)_\sim$, due to the fact that $(\!|w|\!)_\sim a \lhd^* (\!|w'|\!)_\sim b$. Hence $\mathcal{U}_{R_i}(N)|_{\lfloor X \rfloor}$ is acyclic, and is therefore a causal net,

2. $\langle \eta^{R_i}, \beta^{R_i} \rangle : \mathcal{U}_{R_i}(N) \to N$ is a well defined net morphism. Indeed it is routine to check that $\mu\beta^{R_i}(m^{R_i}) = m$, ${}^\bullet\eta^{R_i}(\!|w|\!)_\sim t) = \mu\beta^{R_i}({}^\bullet(\!|w|\!)_\sim t)$ and $\eta^{R_i}(\!|w|\!)_\sim t)^\bullet = \mu\beta^{R_i}(\!|w|\!)_\sim t^\bullet)$, and
3. let $X \in \mathcal{X}(N)$. Consider the firing sequence $m\,[t_1\rangle\, m_1 \ldots [t_n\rangle\, m_X$. We prove that there exists a state $Y \in \mathcal{X}(\mathcal{U}_{R_i}(N))$ such that $\mathcal{P}^{R_i}(Y)$, $\mu\beta^{R_i}(m_Y) = m_X$, and there exists a total order on the elements in Y. The proof is by induction on the length of this firing sequence leading to m_X. If $X = \emptyset$ then $Y = \emptyset$. Assume it holds for $n-1$. Then $X' = X - t_n$ is a state and by induction there is a state $Y' \in \mathcal{X}(\mathcal{U}_{R_i}(N))$ such that $\mu\beta^{R_i}(m_{Y'}) = m_{X'}$ and $\mathcal{P}^{R_i}(Y')$. Take t_n and ${}^\bullet t_n$. Consider the trace $\Downarrow_{\lhd^*} [\![Y']\!] = t_1 \cdots t_{n-1}$ and $s \in {}^\bullet t_n$. The place s is marked at $m_{X'}$ hence there is a place, say (s,i) which is marked at $m_{Y'}$ and $\beta^{R_i}((s,i)) = s$. Clearly $(s,i)^\bullet = \emptyset$ in the subnet $\mathcal{U}_{R_i}(N)|_{[\![Y']\!]}$. Take $t = (\!|\, \|t_1 \cdots t_{n-1}\|_{\mathcal{C}(t_n)} \,|\!)_\sim \cdot t_n$, make it greater of all the transitions in Y', and consider $Y = Y' + t$. Clearly Y is a state of $\mathcal{U}_{R_i}(N)$, it holds that $\mathcal{P}^{R_i}(Y)$ and $\mu\beta^{R_i}(m_Y) = m_X$.

R_c-unfolding: We can now introduce the second of the two new notions of unfolding of a net, which will turn out again to be an unravel net, and which will denote with $\mathcal{U}_{R_c}$. In the I-unfolding each event codes information on its complete *individual* history, whereas in the R_i-unfolding an event codes all the histories with the same Parikh vector with respect to the neighborhood of the corresponding transition. In this unfolding each event represents the i-th occurrence of the corresponding transition, equating also *alternative* histories.

The places of this unfolding are the same of the previous one, thus the only problem is to identify the transitions.

Consider the unfolding in Fig. 4. The two transitions ba and ca correspond to the same occurrence of the transition a in the original net, as the two past histories can be considered exactly the same from the point of view of the produced effects (i.e., the token in q). We recall that, with respect to the situation depicted in Fig. 3, the i-th occurrence of b consumes a token produced by x or y, and the i-th occurrence of x produces the j-th token in the place s when the number of happening of y, a, b and c are in a precise relation, which we have discussed before. Consider now the situation depicted in Fig. 5. The transition x puts tokens in places s and s', whereas the transition y just put tokens in the place s. The i-th firing of the transition x puts the j-th token in place s and the k-th token in the place s', and in general $k \neq j$. In the R_i-unfolding this was easily accomplished just by counting the occurrences of x and y in the preset of s and of s'. In the R_c-unfolding we indicate explicitly in which places the j-th and k-th tokens are produced. Summing up, an event must have the information on the number of its occurrence and on which occurrence of token is producing in a place. Given a safe net $N = \langle S, T, F, m \rangle$, we assume that S is suitably ordered and with $|S|$ we indicate its cardinality. Transition are then triples $(t, i, \alpha) \in T \times \mathbb{N}^+ \times (\mathbb{N}^+)^{|S|}$, where i indicates the number of occurrences and α is a tuple where the positions corresponding to the places in the postset of t have

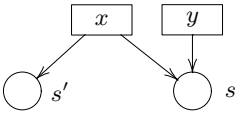


Fig. 5. Part of a safe net

the information on the token produced in these places and 0 otherwise. Transitions and places are connected as previously: the transition (t, i, α) can consume the token in place (s, k), provided that $k = i$, and it produces a token in the place (s', j), with $\alpha(s') = j$, assuming that $F_{pre}(t, s)$ and $F_{post}(t, s')$. Concerning the property, we simply require that if the i-th occurrence of a transition t is in a state, then also the previous $i - 1$ occurrences must be present, and that the tuples α can be suitably ordered. Given a sequence $\overline{\alpha} = \alpha_1, \ldots, \alpha_n$, with $\overline{\alpha}(s)$ we indicate the sequence of numbers $\alpha_1(s), \ldots, \alpha_n(s)$. We say that $\overline{\alpha} = \alpha_1, \ldots, \alpha_n$ is *complete* if each sequence of numbers obtained by $\overline{\alpha}(s)$ deleting all zeros, denoted with $\mathfrak{s}(\overline{\alpha}(s))$, is either empty, strictly increasing and without gaps, where the latter means that

- $\mathfrak{s}(\overline{\alpha}(s))_i + 1 = \mathfrak{s}(\overline{\alpha}(s))_{i+1}$,
- $\mathfrak{s}(\overline{\alpha}(s))_1 = 1$ if the place is initially unmarked in the original net, and
- $\mathfrak{s}(\overline{\alpha}(s))_1 = 2$ if the place is initially marked in the original net.

Finally let $w \in (T \times \mathbb{N}^+ \times (\mathbb{N}^+)^{|S|})^*$, with α_w we denote the sequence of α associated to w.

Definition 11. *Let $N = \langle S, T, F, m\rangle$ be a safe net. The R_c-unfolding $\mathcal{U}_{R_c}(N) = (\langle S^{R_c}, T^{R_c}, F^{R_c}, m^{R_c}\rangle, \mathcal{P}^{R_c})$ is the net defined as follows:*

$$S^{R_c} = (S \times \mathbb{N}^+) \cup (T^{R_c} \times \{*\})$$

$$T^{R_c} = T \times \mathbb{N}^+ \times (\mathbb{N}^+)^{|S|}$$

$$F^{R_c} = \begin{cases} F^{R_c}_{pre}((t, i, \alpha), (s, k)) \text{ iff } F_{pre}(t, s),\ i \leq k \text{ and } k = \alpha(s) \\ F^{R_c}_{pre}((t, i, \alpha), ((t, i, \alpha), *)) \\ F^{R_c}_{post}((t, i, \alpha), (s, k)) \text{ iff } F_{post}(t, s),\ \text{and } k = \alpha(s) + m^{R_c}(s, 1) + 1 \end{cases}$$

$$m^{R_c} = \{((s, 1)) \mid m(s) > 0\} \bigcup \{(e, *) \mid e \in T^{R_c}\}$$

$\mathcal{P}^{R_c}(X)$ *iff X is a state and for all $(t, i, \alpha) \in [\![X]\!]$, if $i > 1$ then $(t, n, \alpha') \in [\![X]\!]$ for all $n < i$, and let a trace w leading to m_X, the sequence of $\mathfrak{s}(\overline{\alpha_w}(s))$ is complete.*

Furthermore $\eta^{R_c} : T^{R_c} \to T$ is defined as $\eta((t, i, \alpha)) = t$ and $\beta^{R_c} : S^{R_c} \to S$ is a multirelation defined as $\beta^{R_c}((s, j)) = s$ iff $s \in S$.

Differently from the R_i-unfolding, the transitions representing the firing of the same transition of the original net and with the same preset and postset in the R_i-unfolding, are now identified. With respect to the previous notion, the two transitions *ca* and *ba*, corresponding to the first occurrence of the transition *a* in the net N, are now identified in the unfolding in Fig. 6. Observe that these two transitions, in the unfolding in Fig. 4, have the same preset and postset. We can prove a result analogous to the proposition 5 simply by observing that the transitions that are now glued have the same preset and postset in the R_i-unfolding.

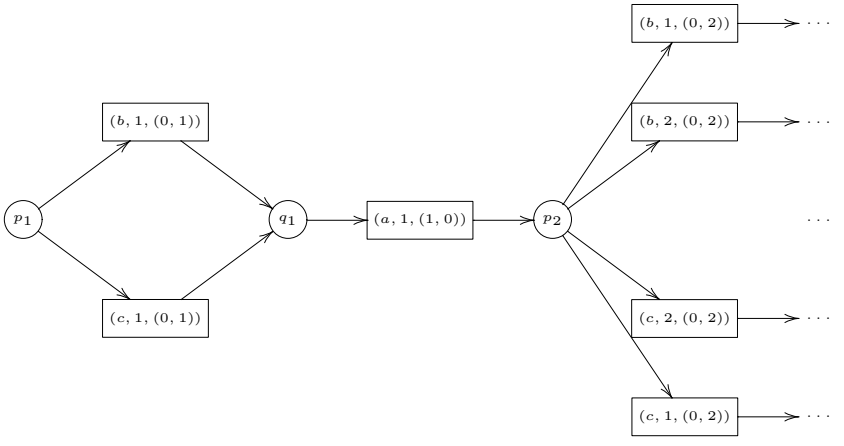


Fig. 6. Part of the R_c-unfolding of N (places $(t, *)$ are omitted)

Proposition 6. *Let $N = \langle S, T, F, m\rangle$ be a safe net and $\mathcal{U}_{R_c}(N) = (\langle S^{R_c}, T^{R_c}, F^{R_c}, m^{R_c}\rangle, \mathcal{P}^{R_c})$ its R_c-unfolding. Then*

1. *$\mathcal{U}_{R_c}(N)$ is an unravel net,*
2. *$\langle \eta^{R_c}, \beta^{R_c}\rangle : \mathcal{U}_{R_c}(N) \to N$ is a well defined net morphism, and*
3. *for each state $X \in \mathcal{X}(N)$ there exists a state $Y \in \mathcal{X}(\mathcal{U}_{R_c}(N))$ such that $\mathcal{P}R_c(Y)$ and $\mu\beta^{R_i}(m_Y) = m_X$.*

Proof. $\mathcal{U}_{R_c}(N)$ is clearly an unravel net, and $\langle \eta^{R_c}, \beta^{R_c}\rangle : \mathcal{U}_{R_c}(N) \to N$ is a well defined net morphism.

Assume the X is a state in $\mathcal{X}(N)$, and consider a firing sequence associated to it. If $X = \emptyset$ then $Y = \emptyset$ is a well defined state. Take $m\,[t_1\rangle\, m_1 \cdots m_{X-t_n}\,[t_n\rangle\, m_x$. By induction to $X - t_n$ a state Y' corresponds and to Y' the firing sequence $m^{R_c}\,[(t_1, 1, \alpha)\rangle\, \hat{m}_1 \ldots m_{Y'}$. Consider $\hat{m}_{i-1}\,[(t_i, k, \alpha_i)\rangle\, \hat{m}_i$, and take $\alpha_i(s) = k$ and $k \neq 0$. Then $\hat{m}_i(s, k)$ is marked. Clearly (t_n, k, α_n) where $k - 1$ is the number of occurrences of t_n in the firing sequence associated to X, and $\alpha_n(s) = l$ if either $s \notin t^\bullet$ and $m_{Y'}(s, i)$ or the place (s, l) does not belong to $\mathcal{U}_{R_c}(N)|_{[\![Y']\!]}$ but $(s, l-1)$ does, is the transition we can add to Y' obtaining the required Y. Obviously Y satisfies the property $\mathcal{P}^{R_c}$.

Collective unfolding: In the case of the collective token philosophy we adopt a slightly different notion, as we associate to the unfolding also a folding morphism.

Definition 12. *Let $N = \langle S, T, F, m\rangle$ be a safe net. The C-unfolding $\mathcal{U}_C(N) = \langle S^C, T^C, F^C, m^C\rangle$ of the net $N = \langle S, T, F, m\rangle$ is the net defined as follows:*

$$S^C = S \cup (T^C \times \{*\})$$
$$T^C = T \times \mathbb{N}^+$$

$$F^C = F^C_{pre}((t,i),s) \text{ if } F_{pre}(t,s) \text{ or } s = ((t,i),*)$$
$$F^C_{post}((t,i),s) \text{ if } F_{post}(t,s)$$
$$m^C = \{s \mid m(s) > 0\} \bigcup \{(e,*) \mid e \in T^C \text{ and} i \in \mathbb{N}^+\}$$

Furthermore $\eta^C : T^C \to T$ *is defined as* $\eta(t,j) = t$ *and* $\beta^C : S^C \to S$ *is a multirelation defined as* $\beta(s) = s$ *iff* $s \in S$

The intuition behind this construction is quite simple: we introduce as many copies of the same transition (which are all distinct because of the index) and we guarantee that they can fire just once. The other connections are just inherited by the original net. Clearly the whole history is lost.

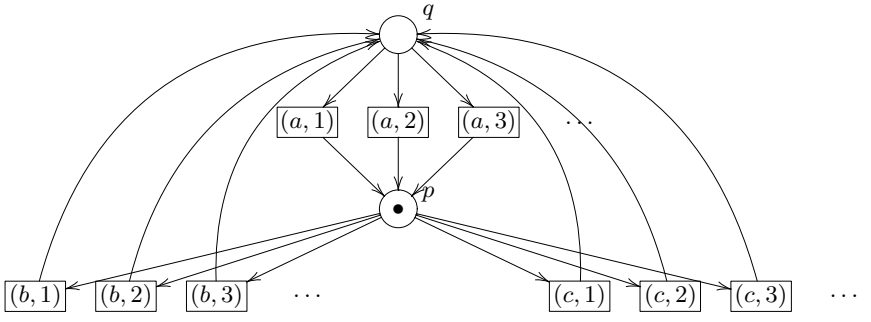


Fig. 7. Part of the C-unfolding of N (places $(t,*)$ are omitted)

Proposition 7. *Let* N *be a safe net and* $\mathcal{U}_C(N)$ *its* C-unfolding. *Then* $\mathcal{U}_C(N)$ *is an occurrence net and* $\langle \eta^C, \beta^C \rangle : \mathcal{U}_C(N) \to N$ *is a well defined net morphism.*

It is straightforward to observe that to a state $X \in \mathcal{X}(N)$ can be easily associated a state Y in $\mathcal{X}(\mathcal{U}_C(N))$, by simply adding different indexes to occurrences of the transitions in X.

4 Relating the Notions of Unfolding

In this section we relate the notions introduced so far. We first show how the I and the R_i-unfoldings are related. Let $N = \langle S, T, F, m \rangle$ be a safe net and $\mathcal{U}^I(N) = \langle B, E, F, m \rangle$ be its I-unfolding. With $\lceil (Y,s) \rceil_s$ we denote the number of conditions (Y',s) *preceedings* the condition (Y,s) in $\mathcal{U}^I(N)$ (including (Y,s)). Consider now an event (X,t) and a marking m' in $\mathcal{U}^I(N)$ such that $m'[(X,t)\rangle$. To m' corresponds the marking $\beta^{R_i}(m')$, and in particular a firing sequence. A trace z leading to (X,t) is then obtained by this firing sequence. Though there can be more then one, as we are interested in the transitions in the neighborhood of t, all the traces can be considered as equivalent, as soon as they have the same number of occurrences of the transitions in the neighborhood of the transition corresponding to the event (X,t).

Theorem 1. *Let* $N = \langle S, T, F, m \rangle$ *be a safe net. Then there exists a folding morphism* $\langle \eta, \beta \rangle : \mathcal{U}^I(N) \to \mathcal{U}^{R_i}(N)$.

Proof. We construct the morphism as follows: $\beta((Y,s),(s,i))$ iff $\lceil (Y,s) \rceil_s = i$ and $\beta((\{e\},*),(\eta(e),*))$, and $\eta(X,t) = (\!|w|\!)_\sim t$ where $w = \|z\|_{\mathcal{C}(t)}$ and z is a trace leading to (X,t) in $\mathcal{U}^I(N)$. It is routine to check that $\langle \eta, \beta \rangle$ is a well defined net morphism.

The second result is the one relating the two new unfoldings we have introduced.

Theorem 2. *Let $N = \langle S,T,F,m \rangle$ be a safe net. Then there exists a folding morphism $\langle \eta, \beta \rangle : \mathcal{U}^{R_i}(N) \to \mathcal{U}^{R_c}(N)$.*

Proof. We construct the morphism as follows: β is the identity multirelation on places (s,i) and and $\beta((\{e\},*),(\eta(e),*))$ otherwise, and $\eta((\!|w|\!)_\sim t) = (t,i,\alpha)$ where $i = [w]_t$ and $\alpha(s) = [w]_{\bullet s}$. Clearly $\mu\beta({}^\bullet(\!|w|\!)_\sim t) = {}^\bullet\eta((\!|w|\!)_\sim t)$ and $\mu\beta((\!|w|\!)_\sim t^\bullet) = \eta((\!|w|\!)_\sim t)^\bullet$.

Finally we relate the R_c-unfolding to the C-unfolding.

Theorem 3. *Let $N = \langle S,T,F,m \rangle$ be a safe net. Then there exists a folding morphism $\langle \eta, \beta \rangle : \mathcal{U}^{R_c}(N) \to \mathcal{U}^{C}(N)$.*

Proof. We construct the morphism as follows: $\beta((s,i),s)$, $\beta((\{e\},*),(\eta(e),*))$, and $\eta((t,i,\alpha)) = (t,i)$. It is clearly a well defined morphism.

The relation between the collective unfolding of a net and the individual one is more subtle, as we have to guess a history to be able to relate the collective unfolding to the individual one. If we consider a state of the collective unfolding, we can *unfold* it onto the individual one. Thus we can fix an order in the execution of different occurrences of the same transition (which is arbitrary) and also keep track of the dependencies among different transitions by exploiting which tokens are produced and consumed (for a proof, see [14]).

Theorem 4. *Let $N = \langle S,T,F,m \rangle$ be a Petri net and let $\mathcal{U}_I(N) = \langle B^I, E^I, F^I, m^I \rangle$ and $\mathcal{U}_C(N) = \langle S^C, T^C, F^C, m^C \rangle$ be the individual (resp. collective) unfolding of N. Let X be a state of $\mathcal{U}_C(N)$ and m_X be the reached marking. Then there exists a finite subnet of $N' = \langle B', E', F', m' \rangle$ of $\mathcal{U}_I(N)$ such that*

1. *each reachable marking of N' is a marking of a firing sequence leading to m_X, and*
2. *there exists an one to one correspondence between E' and X.*

Furthermore N' is a causal net such that $|b^\bullet| \leq 1$ for all $b \in B'$.

5 How Much of the History Can be Forgotten?

Before discussing the results presented in this paper, we compare our constructions with the one introduced in [11].

The occurrence depth of a condition c in C is the maximum number of elements with the same label that are on a path from an initial condition to c (a path is just the set of totally ordered elements less then c). A merged process of a safe net is defined as follows:

Definition 13. *Let $\mathcal{U}_I(N) = \langle B^I, E^I, F^I, m^I \rangle$ be the I-unfolding of the net $N = \langle S, T, F, m \rangle$. Then $\mathfrak{M}(\mathcal{U}_I(N))$ is the unravel net obtained with the following two steps:*

Step 1: *the places of $\mathfrak{M}(\mathcal{U}_I(N))$ are obtained by fusing together all the conditions of $\mathcal{U}_I(N)$ which have the same labels and occurrence-depths; each* mp-*condition inherits its label and arcs from the fused conditions, and its initial marking is the total number of minimal conditions which were fused into it, and*

Step 2: *the transitions of $\mathfrak{M}(\mathcal{U}_I(N))$, called* mp-*events, are obtained by merging all the events which have the same labels, presets and postsets (after step 1 was performed); each mp-event inherits its label from the merged events (and has exactly the same connectivity as either of them).*

Certainly both our unfoldings represent places in the same way, but there are differences in the transitions, as the merged process may identify transitions corresponding to different occurrences of the same transitions. Consider the unfolding in Fig. 2 and apply to the causal net the construction of definition 13. We obtain the following: where the two occurrence of b after the first a are

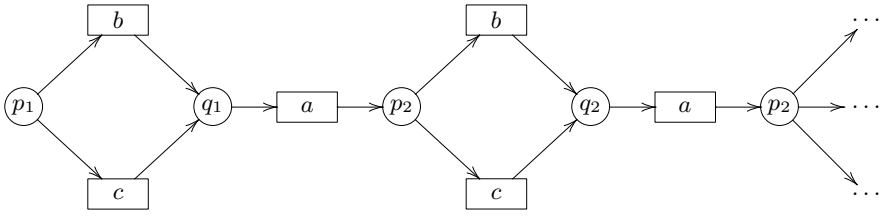


Fig. 8. Part of the merged process of N (places $(t, *)$ are omitted)

identified. Thus a merged process is a more compact representation of the unfolding, though retrieving which is actually the correct run of the system may be less direct with respect to the approach taken here. The information which is abstracted away in the merged processes view with respect to the R_c-unfolding is the *index* of the occurrence of the transition, thus there exists a mapping from the R_c-unfolding to the merged process of a net N. However this relation does not fit in the view of this paper as there is no obvious relation between merged processes and collective unfolding. In fact our target was different with respect to the one of [11], as we have focussed our attention on how much of the history can be kept by direct constructions. To this aim we have introduced two new notions of unfolding, obtaining a more compact and possibly useful representation of the non sequential behaviour of a safe net.

Let us sum up the results. The taxonomy emerging from these notions is the following:

– the individual unfolding models the whole history of each token and happening of a transition. As the whole history is kept, the same information

is spread in many points of the unfolding, and this information could be represented more compactly;
- the first step toward a more compact representation, still being able to reconstruct easily the whole history, is represented by the R_i-unfolding, where the i-th occurrences of a token in a place are equated. In this unfolding a transition keeps its local history, namely the counts of the occurrences of transitions in its neighborhood;
- the second step is to identify the happening of the transitions that, though may have different histories, are from the point of view of external observation, undistinguishable: they produce the same tokens and correspond to the same occurrence of the transition (i.e., the case of the i-th firing of a transition). Nevertheless some relevant dependencies are kept, as it is still possible to look at the unfolding and retrieve a part of the history;
- finally the history can be totally forgotten, in the collective unfolding. In this case the only way to find out the dependencies among occurrences of different transitions is to look at all the possible executions.

The relations among these notions show that the *fading* of the history can be modeled. To obtain these results we have formalized a notion of unravel net, where the original net is not completely unfolded, but rather somehow *unraveled*.

Concerning to the question '*How much of the history can be forgotten*', the answer obviously depend on the use of the history one want to realize. Here we have shown that there are construction in between the two extrema of what it seems to be a whole spectrum. The problem of retrieving a net from an unfolding has not be pursued in this paper, but the classical approaches do applies to the unfoldings presented, provided that the property is kept into account.

Though we have not addressed the problem of the unfolding in the case of non safe nets, our constructions can be easily lifted to take into account these nets, observing that the relations among indexes are much more complex. Furthermore the property characterizing unravel nets may be presented using suitable combinations of relations which can be extracted from the net itself, as it is done in causal net.

Many relevant issues have not been addressed in this paper. We just point out two of them. One concerns the relation of these unfoldings with event structures, and the other regards the categorical investigation of these constructions. These will be the subject of further investigations.

Acknowledgement. I would like to thanks the reviewer for their useful comments and suggestions.

References

1. Winskel, G.: Event Structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)
2. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica 28, 575–591 (1991)

3. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri nets, asymmetric event structures and processes. Information and Computation 171, 1–49 (2001)
4. Baldan, P., Busi, N., Corradini, A., Pinna, G.: Domain and event structure semantics for Petri nets with read and inhibitor arcs. Theoretical Computer Science 323, 129–189 (2004)
5. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures. In: Kozen, D. (ed.) Proceedings of 10^{th} Annual IEEE Symposium on Logic in Computer Science, pp. 199–209. IEEE Computer Society Press, Los Alamitos (1995)
6. Gunawardena, J.: A generalized event structure for the Muller unfolding of a safe net. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 278–292. Springer, Heidelberg (1993)
7. Gunawardena, J.: Geometric logic, causality and event structures. In: Groote, J.F., Baeten, J.C.M. (eds.) CONCUR 1991. LNCS, vol. 527, pp. 266–280. Springer, Heidelberg (1991)
8. Gunawardena, J.: Causal automata. Theoretical Computer Science 101, 265–288 (1992)
9. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theoretical Computer Science 153, 129–170 (1996)
10. Lorenz, R., Juhás, G., Bergenthum, R., Desel, J., Mauser, S.: Executability of scenarios in Petri nets. Theoretical Computer Science 410, 1190–1216 (2009)
11. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged processes: a new condensed representation of Petri net behaviour. Acta Informatica 43, 307–330 (2006)
12. Fabre, E.: Trellis processes: A compact representation for runs of concurrent systems. Discrete Event Dynamic Systems 17, 267–306 (2007)
13. van Glabbeek, R.J.: The individual and collective token interpretations of petri nets. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 323–337. Springer, Heidelberg (2005)
14. Pinna, G.M.: Petri nets unfoldings and the individual/collective token philosophy. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 279–288. Springer, Heidelberg (2009)
15. Reisig, W.: Petri Nets: An Introduction. EACTS Monographs on Theoretical Computer Science. Springer, Heidelberg (1985)
16. Hayman, J., Winskel, G.: The unfolding of general Petri nets. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008),, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2008)

Branching Processes of General Petri Nets*

Jean-Michel Couvreur[1], Denis Poitrenaud[2], and Pascal Weil[3]

[1] LIFO, Université d'Orléans, Orléans, France
Jean-Michel.Couvreur@univ-orleans.fr
[2] LIP6, Université Pierre et Marie Curie, Paris, France
Denis.Poitrenaud@lip6.fr
[3] LaBRI, Université de Bordeaux I, Talence, France
Pascal.Weil@labri.fr

Abstract. We propose a new model of branching processes, suitable for describing the behavior of general Petri nets, without any finiteness or safeness assumption. In this framework, we define a new class of branching processes and unfoldings of a net N, which we call true. These coincide with the safe branching processes and unfoldings if N is safe, or weakly safe as in [Engelfriet 1991], but not in general. However, true branching processes and processes satisfy the good order-theoretic properties which make the safe processes of safe nets so useful in practice, and which are known not to hold for the safe processes of a general net. True processes represent therefore good candidates to generalize the theory of safe nets to the general case.

1 Introduction

The study of the behavior of models of concurrency usually requires the definition of more abstract models. Within the framework of Petri nets, primarily two models were retained: labeled occurrence nets and event structures. Both models were proposed by Nielsen, Plotkin and Winskel in their foundational paper [7], in order to give a semantic of concurrency for safe Petri nets. The description of a safe Petri net execution is presented by a labeled causal net, called a process. Roughly speaking, causal nets are acyclic Petri nets whose places are without branching. In particular, their places and transitions are partially ordered, and this order, restricted to transitions, induces a partial order on the transition occurrences in the original Petri net. For the representation of conflictual behaviors, branching on places is authorized. This leads to the definition of labeled occurrence nets, called branching processes. The set of all the behaviors of the system can be captured by a single branching process, called the unfolding of the system, whose transitions are called events. By restricting the relations of causality and conflict to events, one obtains an event structure called the prime event structure. Since the publication of [7], there has been many attempts to extend these results to general Petri nets.

* This paper was prepared in part while the first author was on a 2-year assignment at LSV (ENS Cachan) with the support of CNRS. The third author acknowledges partial support from Project VERSYDIS, ACI *Sécurité informatique*, Ministère de l'Industrie.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 129–148, 2011.

In these attempts, the focus has often been on using the same classes as in the case of safe nets, namely causal and occurrence Petri nets. Engelfriet [2] restricted his work to 1-valued, initially 1-marked Petri nets, and he obtained good algebraic properties on branching processes (a structure of lattice) which led to the concept of unfolding.

For others (e.g. Best and Devillers [1], Meseguer, Montanari and Sassone [5]), tokens are individualized. As argued in [4], this is contrary to a pure multiset view of general nets, and the systems modelled by Petri nets rarely justify individualizing tokens. Similarly, Haar [3], pursuing Vogler's work [9], proposes an approach which aims at translating general nets into safe nets, by introducing a place for each reachable marking of original places. This not only considerably increases the size of the structure, but it also artificially introduces conflicts between transitions that access a given place. Thus it strongly departs from the intended semantic of nets.

In contrast with these approaches, Hoogers, Kleijn and Thiagarajan [4], propose a new event structure. In this so-called local event structure, tokens are not colored. Their theory is complete for co-safe nets (see [4]), and it can be extended to the case of general nets. However, it does not present the expected properties in the general case.

In this paper, we propose a more net-theoretic approach, which does not impose to individualize the tokens, and which incorporates the solutions of [1,2,5]. Our formal framework allows us to identify a new structure, called true unfolding, which allows for the good algebraic properties identified by Engelfriet [2], and which is applicable to general nets.

The starting point of our approach is an extension of the definitions of occurrence nets, which can be arbitrarily valued and non-safe, and of configurations, which are multisets of transitions. Branching processes are defined as occurrence nets labeled by the elements of the original net. The set of branching processes of a net is equipped with a natural order relation, which leads to the definition of unfoldings as maximal branching processes.

We identify two classes of branching processes and two types of unfoldings: safe and true. The safe case coincides with those of [1,2,5]. The true case is more interesting because of its order-theoretic properties. For safe nets, and more generally under the constraints on the net structure identified by Engelfriet in [2], there exists a unique unfolding, and the concepts of safe and true unfolding coincide. In general however, the two unfoldings differ.

We also formalize the concept of process in our multiset context. Contrary to other works, our definition is not based on causal nets, but on the concept of a configuration. Again, it turns out that the safe case does not offer good enough properties (*e.g.*, we cannot define the greatest lower bound of two processes). In contrast, the expected properties hold for true processes. This comes from the fact that a true process is represented in a unique way in the true unfolding.

The drawback of these algebraically and order-theoretically satisfactory structures is that the concepts of conflict and causality are not any more explicitly given by the model structure. That is, we lose a direct link with prime event structures, as in [4].

The paper is organized as follows. In Section 2, we fix the notation for nets, homomorphisms of nets and other fundamental objects. Section 3 discusses occurrence nets and configurations. In Section 4, generalized branching processes and unfoldings

are introduced, and the properties of true and safe branching processes and unfoldings are compared. Finally Section 5 presents the generalized notion of processes, and establishes the order-theoretic properties of true processes. Counter-examples of these properties for safe processes are also presented.

2 Preliminaries

We first summarize the basic notation and concepts used in this paper, concerning multisets and Petri nets. denotes the set of non-negative integers.

2.1 Notation

Let X be a set. A *multiset* over X is a mapping $v: X \to$. Multisets are often represented as formal linear combinations, *e.g.* $v = a + 2b$ for $v(a) = 1$, $v(b) = 2$, $v(c) = 0$ for all $c \in X \setminus \{a, b\}$, and it is also convenient to view them as vectors in ^{X}. The *support* of the multiset v is the set $\overline{v} = \{x \in X \mid v(x) > 0\}$. Note that the support of a multiset may be infinite. A multiset over a set X can be naturally considered as a multiset over any superset of its support.

The operations of addition and subtraction of multisets over X are defined componentwise, as on vectors (note however that negative coefficients are not allowed). An infinite sum of multisets $v = \sum_{i \in I} v_i$ is said to be *well-defined* if for each $x \in X$, the sum $\sum_{i \in I} v_i(x)$ is finite. If $\sigma = x_1 x_2 \cdots$ is a finite sequence of elements of X, the *characteristic vector* of σ is the multiset $\sigma = \sum_{i=1}^{|\sigma|} x_i$. Multisets are partially ordered by letting $v \leq w$ if $v(x) \leq w(x)$ for each $x \in X$.

If X and Y are sets, a mapping $h: X \to Y$ can be partially extended to multisets, $h:\ ^X \to\ ^Y$, by letting $h(v) = \sum_{x \in X} v(x)h(x)$ if the sum is well-defined (that is, if each $y \in Y$ has finitely many pre-images in the support of v). The mapping h can also be extended to sequences of elements of X by letting $h(x_1 x_2 \cdots) = h(x_1)h(x_2)\cdots$.

Let $\to$ be a relation on the set X, for instance the edge relation of a graph in which X is the set of vertices. We denote by $\stackrel{*}{\to}$ (resp. $\stackrel{+}{\to}$), the reflexive and transitive closure (resp. transitive) of $\to$. We also use the following notation: if $Y \subseteq X$,

$$
\begin{aligned}
{}^\bullet Y &= \{x \in X \mid \exists y \in Y,\ x \to y\} & {}^* Y &= \{x \in X \mid \exists y \in Y,\ x \stackrel{*}{\to} y\} \\
Y^\bullet &= \{x \in X \mid \exists y \in Y,\ y \to x\} & Y^* &= \{x \in X \mid \exists y \in Y,\ y \stackrel{*}{\to} x\} \ .
\end{aligned}
$$

When the graph $(X, \to)$ is *acyclic* (i.e, $x \stackrel{+}{\to} x$ never holds, for any $x \in X$), the relation $\stackrel{*}{\to}$ forms a partial order on a set X.

Finally, if $(X, \leq)$ is a partially ordered set and if $Y \subseteq X$, we say that x is a *lower* (resp. *upper*) *bound* of Y if $x \leq y$ (resp. $y \leq x$) for each element $y \in Y$. The *greatest lower bound* (resp. *least upper bound*) of Y, if it exists, is denoted by $\inf(Y)$ (resp. $\sup(Y)$). If any two elements of X admit a lower bound and an upper bound, X is called a *lattice*. It is a *complete lattice* if any subset of X admits a lower and an upper bound.

2.2 Petri Net

A *Petri net*, or simply a *net*, is a tuple $N = (P, T, \mathsf{Pre}, \mathsf{Post}, m_0)$ consisting of two disjoint sets P and T whose elements are called *places* and *transitions*, two multisets Pre and Post over $P \times T$ (sometimes called the *flow functions*), and a multiset m_0 over P called the *initial marking*. A *marking* of N is any multiset over P. If t is a transition, the *pre-condition* of t, written $\mathsf{Pre}(t)$, is the marking $\mathsf{Pre}(\cdot, t)$. Similarly, the *post-condition* of t, written $\mathsf{Post}(t)$ is the marking $\mathsf{Post}(\cdot, t)$.

Note that we don't make any finiteness assumption: P or T may be infinite, as well as the support of the initial marking. A net may be viewed as a labelled bipartite graph (the graphical representation of N) as follows: we can identify N with the labeled graph $(P, T, \rightarrow, m_0)$, where places and transitions are the two disjoint sets of nodes, there is an edge $p \xrightarrow{\mathsf{Pre}(p,t)} t$ (resp. $t \xrightarrow{\mathsf{Post}(p,t)} p$) between the place p and the transition t if $\mathsf{Pre}(p,t)$ (resp. $\mathsf{Post}(p,t)$) is non-zero, and m_0 is a labelling function of the places, traditionally depicted by the presence of $m_0(p)$ tokens in place p.

If t is a transition and m is a marking of N, we say that t is *enabled* by m, written $m\,[t\rangle$, if $\mathsf{Pre}(t) \leq m$. *Firing the transition* t in m produces the marking

$$m' = m + \mathsf{Post}(t) - \mathsf{Pre}(t) \quad . \qquad \textbf{(firing rule)}$$

and we write $m\,[t\rangle\, m'$. The firing rule is extended inductively to any sequence of transitions: if ε is the empty sequence, we let $m\,[\varepsilon\rangle\, m$; if σ is a sequence of transitions and if t is a transition, then $m\,[\sigma\, t\rangle\, m'$ if there exists a marking m'' such that $m\,[\sigma\rangle\, m''$ and $m''\,[t\rangle\, m'$. It is easily verified that if $\sigma = t_1 t_2 \cdots t_n$ and $m\,[\sigma\rangle\, m'$, then

$$m' = m + \sum_{i=1}^{n} \mathsf{Post}(t_i) - \sum_{i=1}^{n} \mathsf{Pre}(t_i) \quad . \qquad \textbf{(extended firing rule)}.$$

A transition, or sequence of transitions, is said to be *firable* if it is enabled by the initial marking. A marking m is called *reachable* if there exists a finite sequence of transitions σ such that $m_0\,[\sigma\rangle\, m$. We denote by $\mathsf{Reach}(N)$ the set of reachable markings of N.

A transition t is called *spontaneous* if $\mathsf{Pre}(t) = 0$, that is, ${}^\bullet t = \emptyset$. A place or a transition x is called *isolated* if ${}^\bullet x = x^\bullet = \emptyset$.

In the sequel, we will discuss a number of properties of nets. Recall that a marking m is *safe* if $m(p) \leq 1$ for each place p. The net N is said to be

- **finite** if P and T are finite sets;
- **elementary** if the pre- and post-condition $\mathsf{Pre}(t)$ and $\mathsf{Post}(t)$ of each transition t are safe markings;
- **weakly safe** if N is elementary, m_0 is safe and every spontaneous transition is isolated;
- **safe** if each reachable marking is safe;
- **quasi-live** if every transition is enabled by a reachable marking;
- **acyclic** if the graph representing N is acyclic; in that case, the induced partial order on the set of places and transitions is written $\leq$.

Most of these notions are classical [6], except for weakly safe nets, which are new. They occur nevertheless in Engelfriet [2].

Lemma 1. *Every quasi-live, safe Petri net is weakly safe.*

It is interesting to note that weak safeness is a property that can be easily verified upon reading the definition of a Petri net, whereas deciding safeness requires computing a transitive closure.

For further reference, we note the following technical lemma, which belongs to the folklore.

Lemma 2. *Let N be an acyclic net, let t be a transition, and let σ be a minimal length firable sequence of transitions enabling t. Then every transition x in σ is such that $x < t$.*

2.3 Homomorphism of Nets

Let $N = (P, T, \mathsf{Pre}, \mathsf{Post}, m_0)$ and $N' = (P', T', \mathsf{Pre}', \mathsf{Post}', m'_0)$ be nets. Let $h: P \cup T \to P' \cup T'$ be a mapping such that $h(P) \subseteq P'$ and $h(T) \subseteq T'$. We say that h is a *homomorphism of nets* from N to N' (see [2]) if, for each transition $t \in T$, we have

- $\mathsf{Pre}'(h(t)) = h(\mathsf{Pre}(t))$,
- $\mathsf{Post}'(h(t)) = h(\mathsf{Post}(t))$,
- $m'_0 = h(m_0)$.

Observe that in this definition, $h(m_0)$, $h(\mathsf{Pre}(t))$ and $h(\mathsf{Post}(t))$ must be well-defined, that is, the pre-image $h^{-1}(p')$ of each place $p' \in P'$ must have a finite intersection with the support of m_0 and of each pre- and post-condition of a transition of N (see Section 2.1).

We note the following elementary properties of homomorphisms of nets.

Lemma 3. *Let $N = (P, T, \mathsf{Pre}, \mathsf{Post}, m_0)$ and $N' = (P', T', \mathsf{Pre}', \mathsf{Post}', m'_0)$ be nets, and let $h: N \to N'$ be a homomorphism.*

- *If a transition t of N' sits in the range of h, then ${}^\bullet t$ and $t^\bullet$ are contained in the range of h.*
- *Let $t \in T$. If $\mathsf{Pre}'(h(t))$ (resp. $\mathsf{Post}'(h(t))$) is a safe marking, then the same holds for $\mathsf{Pre}(t)$ (resp. $\mathsf{Post}(t)$) and h is injective on ${}^\bullet t$ (resp. $t^\bullet$).*
- *If m'_0 is safe, then m_0 is safe and h is injective on $\overline{m_0}$.*
- *If N' is elementary (resp. weakly safe), then so is N.*

Proposition 1. *Let N and N' be Petri nets and let $h: N \to N'$ be a homomorphism. Let m be a marking of N such that $h(m)$ is well-defined, let t be a transition of N enabled by m, and let m_1 be the resulting marking, that is, $m\,[t\rangle\, m_1$. Then $h(m_1)$ is well-defined and $h(m)\,[h(t)\rangle\, h(m_1)$.*

Proposition 1 can be extended by induction to sequences of transitions. This yields immediately the following corollary.

Corollary 1. *Let N and N' be Petri nets and let $h: N \to N'$ be a homomorphism. Then $h(\mathsf{Reach}(N)) \subseteq \mathsf{Reach}(N')$. Moreover, if N' is safe, then N is safe.*

2.4 Subnets of Nets

Let $N = (P, T, \mathsf{Pre}, \mathsf{Post}, m_0)$ and $N' = (P', T', \mathsf{Pre}', \mathsf{Post}', m'_0)$ be nets. We say that N' is a *subnet* of N, and we write $N' \sqsubseteq N$, if $P' \subseteq P$, $T' \subseteq T$, $m_0 = m'_0$ and for each transition t of N', $\mathsf{Pre}(t) = \mathsf{Pre}'(t)$ and $\mathsf{Post}(t) = \mathsf{Post}'(t)$.

Observe that this definition is different from the sole requirement that $P' \cup T' \subseteq P \cup T$ and Pre', Post' and m'_0 are the restrictions of Pre, Post and m_0 to the places and transitions of N': consider for instance the case where N and N' have a single transition t, N' has places p' and q', and N has places p, q, p' and q', $m_0 = m'_0 = p'$, $\mathsf{Pre}'(t) = p$, $\mathsf{Pre}(t) = p + p'$, $\mathsf{Post}'(t) = q$, $\mathsf{Post}(t) = q + q'$.

Assuming that the isolated places, if there are any, are in the initial marking, a subnet N' of a net N is entirely determined by the set T' of its transitions: indeed, we have necessarily $P' = \overline{m_0} \cup \bigcup_{t \in T'} ({}^\bullet t \cup t^\bullet)$ and the pre- and post-conditions of the transitions of N' are the same as in N. Moreover, every subset of transitions of N gives rise to a subnet of N.

If N is an acyclic net, we say that a subnet N' is a prefix of N if $P' \cup T'$ is an $\leq$-order ideal of $P \cup T$. We note the following simple observations.

Lemma 4. *Let N and N' be Petri nets.*

1. *N' is isomorphic to a subnet of N if and only if there exists an injective net homomorphism from N' into N.*
2. *The subnets of N form a complete lattice.*
3. *If N is acyclic, then every subnet of N is acyclic.*

3 Occurrence Nets

A net N is an *occurrence net* if, for each place $p \in P$,

- if $m_0(p) > 0$, then p is an initial place (${}^\bullet p = \emptyset$);
- if $m_0(p) = 0$, then p receives its inputs from a single transition ($|{}^\bullet p| = 1$), denoted by ${}^\bullet p$;
- N is quasi-live (every transition is enabled by some reachable marking).

Occurrence nets were structurally introduced by Nielsen, Plotkin and Winskel [7] in the safe case using concurrency relation over transitions and places. Our definition is somewhat simpler, and it makes no assumption of safeness or finiteness. However, for safe nets, the two definitions coincide (see subsection 3.1).

3.1 Elementary Properties of Occurrence Nets

Proposition 2. *Let $N = (P, T, \mathsf{Pre}, \mathsf{Post}, m_0)$ be an occurrence net. Then N is acyclic and every vertex of the graph N is preceded by a finite number of transitions. That is, if $x \in P \cup T$, then ${}^* x \cap T$ is finite.*

Proof. As the graph (underlying) N is bipartite, if N has a cycle, then there exists a sequence of places $p_0, p_1, \ldots, p_{n-1}$ and a sequence of transitions $t_0, t_1, \ldots, t_{n-1}$ such that ${}^\bullet p_i = t_i$ and $t_i \in p_{i-1}{}^\bullet$ for $i = 0, \ldots, n$ (where i is taken modulo n). The value at p_i of a marking of N can be modified only by firing transition t_i, and transition t_i is enabled by a marking m only if $m(p_{i-1}) \neq 0$. Finally, as none of the places p_i is an initial place, the initial marking m_0 is 0 on each p_i: it is now immediately verified, by induction on the length of a firing sequence, that no reachable marking enables any of the transitions t_i, thus contradicting the assumption that N is quasi-live.

Thus N is acyclic. In particular, the set of vertices of the graph N is partially ordered by the relation $\stackrel{*}{\rightarrow}$.

Let $p \in P$ be a place such that ${}^*p \neq \emptyset$: then ${}^*p \cap T = {}^*({}^\bullet p) \cap T$ by definition of an occurrence net. We now consider the set ${}^*t$, for some transition $t \in T$. Observe that if $t' \in T$ and $p \in P$ are such that $t' \rightarrow p \rightarrow t$ in N, then $t' = {}^\bullet p$ and t can be fired only after t' was fired. This remark is extended by induction to show that if t' is a transition and $t' \stackrel{+}{\rightarrow} t$, then t' must be fired before t can be fired. As an occurrence net is quasi-live, that is, every transition can appear in a finite sequence of transitions enabled by the initial marking, it follows that the set of transitions in ${}^*t$ is finite. □

We note the following property of subnets of occurrence nets. Recall that a subnet N' of a net N is a *prefix* of N if $P' \cup T'$ is an order ideal of $P \cup T$.

Lemma 5. *Let N be an occurrence net and let N' be a subnet of N. Then N' is an occurrence net if and only if N' is a prefix of N.*

Proof. It is immediate that a prefix of an occurrence net is an occurrence net. Conversely, suppose that $N' \sqsubseteq N$. Then $\mathsf{Pre}(t) = \mathsf{Pre}'(t)$ for each transition t of N'. In particular, if t is a transition of N' and $p \rightarrow t$ in N, then $p \in P'$.

If in addition N' is an occurrence net, consider a place p of N'. If $t \rightarrow p$ in N, then $m_0(p) = 0$. If t is not in N', then ${}^\bullet p = \emptyset$ in N', so that $m'_0(p) > 0$, contradicting the equality $m_0 = m'_0$. Thus t is in N' and N' is a prefix of N. □

For occurrence nets, the distinction between safeness and weak safeness vanishes (see Lemma 1).

Proposition 3. *Let N be an occurrence net. Then N is safe if and only if N is weakly safe.*

Proof. In view of Lemma 1, it suffices to show that if the occurrence net N is weakly safe, then it is safe. Let us now assume that N is weakly safe and consider a marking m of N, reached after firing a sequence of transitions $t_1 t_2 \ldots t_n$ from the initial marking m_0. We choose this sequence to be of minimal length, so that it is increasing (Lemma 2 and Proposition 2).

For $1 \leq i \leq n$, let m_i be the marking reached after firing $t_1 \ldots t_i$, that is, $m_0 \ [t_1 \ldots t_i\rangle \ m_i$ and $m_n = m$. In particular $m_i = m_{i-1} - \mathsf{Pre}(t_i) + \mathsf{Post}(t_i)$ for each i.

We verify by induction on i that for each place p such that $m_i(p) > 0$, then either p is an initial place and $m_i(p) = m_0(p)$, or ${}^\bullet p = t_j$ for some (unique) $j \leq i$ and $m_i(p) = m_j(p) = \mathsf{Post}(p, t_j)$. Note that this implies that the marking m_i is safe, since m_0 is safe and N is elementary. Thus this verification will complete the proof of the proposition.

This assertion is trivial if $i = 0$, so we now assume that it is true for some $i < n$ and we consider a place p such that $m_{i+1}(p) > 0$. If $\mathsf{Pre}(p,t_{i+1}) = \mathsf{Post}(p,t_{i+1}) = 0$, then $m_{i+1}(p) = m_i(p)$ and we are done.

If $\mathsf{Pre}(p,t_{i+1}) \neq 0$, then $\mathsf{Post}(p,t_{i+1}) = 0$ by acyclicity and $m_{i+1}(p) = m_i(p) - \mathsf{Pre}(p,t_{i+1})$. Since m_i is safe and N is elementary, it follows that $m_{i+1}(p) \leq 0$, a contradiction.

Finally, if $\mathsf{Post}(p,t_{i+1}) \neq 0$, then ${}^\bullet p = t_{i+1}$ and $m_{i+1}(p) = m_i(p) + \mathsf{Post}(p,t_{i+1})$. If $m_i(p) > 0$, then the induction hypothesis implies that ${}^\bullet p = t_j$ for some $j \leq i$, so $t_{i+1} = t_j$, contradicting the minimality of the firing sequence. Thus $m_i(p) = 0$ and $m_{i+1}(p) = \mathsf{Post}(p,t_{i+1})$, which completes the induction. □

3.2 Configuration

Let N be a Petri net. The characteristic vector of a firable finite sequence of transitions is called a *configuration* of N.

It is not true that, even in an occurrence net, a configuration arises from a unique firable sequence: suppose $P = \{p_1,\ldots,p_4\}$, $T = \{t_1,t_2\}$, $m_0 = p_1 + p_2$, $\mathsf{Pre} = (p_1,t_1) + (p_2,t_2)$ and $\mathsf{Post} = (p_3,t_1) + (p_4,t_2)$. Then t_1t_2 and t_2t_1 are distinct firable sequences yielding the same configuration. More generally, transitions with disjoint pre-conditions can be fired in any order.

It follows however immediately from the extended firing rule that if two firable sequences induce the same configuration φ, then they both lead to the same marking, denoted by $\mathsf{Cut}(\varphi)$. Somewhat abusing definitions, we say that $\mathsf{Cut}(\varphi)$ is the marking reached after firing the vector (or the configuration) φ.

Configurations of acyclic nets are characterized as follows ([6] for the case of finite nets).

Proposition 4. *Let N be a Petri net and let φ be a multiset over T. If φ is a configuration, then the support of φ is finite and, for each place $p \in P$, we have*

$$m_0(p) - \sum_{t\in T} \mathsf{Pre}(p,t)\cdot\varphi(t) + \sum_{t\in T} \mathsf{Post}(p,t)\cdot\varphi(t) \geq 0 \quad . \tag{1}$$

If N is acyclic and φ is a finite support multiset over T satisfying Equation (1) for all places P, then φ is a configuration.

Proof. If φ is a configuration of the Petri net N, then the support of φ is trivially finite, and for each place p, $\mathsf{Cut}(\varphi)(p) \geq 0$: that is exactly the statement in Equation (1).

We now assume that N is acyclic, φ has finite support and Equation (1) holds for each place p. We proceed by induction on the value of $\sum_{t\in T}\varphi(t)$. The empty multiset is certainly a configuration, since it is the sum of the terms of the empty sequence of transitions. Now we assume that $\varphi \neq 0$. The relation $\xrightarrow{*}$ is a partial order by assumption, and we consider a transition s maximal in the support of φ and the multiset ψ such that $\psi + s = \varphi$. Of course, ψ has finite support.

Let $m_\psi(p) = m_0(p) - \sum_{t\in T}\mathsf{Pre}(p,t)\cdot\psi(t) + \sum_{t\in T}\mathsf{Post}(p,t)\cdot\psi(t)$. Then Equation (1) states that

$$m_\psi(p) - \mathsf{Pre}(p,s) + \mathsf{Post}(p,s) \geq 0 \quad .$$

We want to show that $m_\psi(p) \geq 0$ for each place p. If $\mathsf{Post}(p,s) = 0$, then $m_\psi(p) \geq \mathsf{Pre}(p,s) \geq 0$. If $\mathsf{Post}(p,s) \neq 0$, then $s = {}^\bullet p$, and since s is maximal in the support of φ, $\mathsf{Pre}(p,t) = 0$ for each transition t in the support of φ. In particular,

$$m_\psi(p) = m_0(p) + \mathsf{Post}(p,s) \cdot \psi(s) \geq m_0(p) \geq 0 \ .$$

We can now use the induction hypothesis to see that ψ is a configuration – and hence, $m_\psi = \mathsf{Cut}(\psi)$. We already noticed that if $\mathsf{Post}(p,s) = 0$, then $m_\psi(p) \geq \mathsf{Pre}(p,s)$. Moreover, if $\mathsf{Post}(p,s) \neq 0$, then $\mathsf{Pre}(p,s) = 0$ by acyclicity and again, we have $m_\psi(p) \geq \mathsf{Pre}(p,s)$. Thus the transition s is enabled by the marking $\mathsf{Cut}(\psi)$ and $\varphi = \psi + s$ is a configuration. □

In view of the specificities of occurrence nets, Proposition 4 yields the following corollary.

Corollary 2. *Let N be an occurrence net and let φ be a multiset over T. Then φ is a configuration of N if and only if φ has finite support and for each place $p \in P$, we have*

$$\begin{aligned} m_0(p) &\geq \textstyle\sum_{t \in T} \mathsf{Pre}(p,t) \cdot \varphi(t) \quad \textit{if } m_0(p) > 0, \\ \mathsf{Post}(p, {}^\bullet p) \cdot \varphi({}^\bullet p) &\geq \textstyle\sum_{t \in T} \mathsf{Pre}(p,t) \cdot \varphi(t) \quad \textit{otherwise.} \end{aligned}$$

We also note that, as in the safe case, each reachable marking of an occurrence net is reached after firing a uniquely determined configuration.

Proposition 5. *Let N be an occurrence net such that $t^\bullet \neq \emptyset$ for each transition t, and let m be a reachable marking. Then there exists a unique configuration φ of N such that $m = \mathsf{Cut}(\varphi)$.*

Proof. We proceed by induction on the maximal length r of a path in the graph N from an initial place to a place with non-zero m-marking. Since $t^\bullet \neq \emptyset$ for each transition t, the firing of any transition leads to a marking which is non-zero on some non-initial place. Thus, if $r = 0$, then $m = m_0$ and the result is trivial: by acyclicity, m can be reached only if we do not fire any transitions.

In the general case, let φ be a configuration such that $m = \mathsf{Cut}(\varphi)$, and let Q be the set of maximal non-initial places with non-zero m-marking. Then ${}^\bullet Q$ is the set of maximal transitions t such that $\varphi(t) > 0$ and in particular, ${}^\bullet Q$ is finite and $m(q) = \mathsf{Post}(q, {}^\bullet q)\varphi({}^\bullet q)$ for each $q \in Q$. Thus the maximal elements of the support of φ and the value of φ on these elements are uniquely determined by m. Moreover, the maximality of the ${}^\bullet q$ ($q \in Q$) in the support of φ implies that any firable sequence σ whose characteristic vector is equal to φ can be rearranged into a firable sequence of the form $\sigma'\sigma''$ where σ'' contains all the occurrences of the transitions ${}^\bullet q$ ($q \in Q$).

Now let φ' be the characteristic vector of σ' and let

$$m' = m + \sum_{t \in {}^\bullet Q} \mathsf{Pre}(t)\varphi(t) - \sum_{t \in {}^\bullet Q} \mathsf{Post}(t)\varphi(t) \ .$$

Then $m \ [\sigma'\rangle \ m'$, that is, $m' = \mathsf{Cut}(\varphi')$. We note that m' is uniquely determined by m and that it is a reachable marking. Moreover, if p is a maximal non-initial place such that $m'(p) > 0$, then $p < q$ for some $q \in Q$. Thus, by induction, φ' is uniquely determined by m' and hence, by m. Since $\varphi = \varphi' + \sum_{t \in {}^\bullet Q} \varphi(t)t$, this concludes the proof. □

Remark 1. The same uniqueness result holds (with the same proof) for any acyclic net such that $t^\bullet \neq \emptyset$ for each transition t and $|{}^\bullet p| \leq 1$ for each place p.

4 Branching Processes and Unfoldings of a Net

We now discuss the branching processes and the unfoldings of a net [2], within the framework developped in this paper. Bowing to tradition, occurrence nets in branching processes will be usually be written $S = (B, E, \mathsf{In}, \mathsf{Out}, q_0)$, their transitions will be called *events* and their places will be called *conditions*.

4.1 Branching Processes

A *branching process of a net* N is a pair (S,h) consisting of an occurrence net $S = (B, E, \mathsf{In}, \mathsf{Out}, q_0)$ and a homomorphism $h: S \to N$ satisfying a guarded form of injectivity: whenever e and e' are events of S,

$$\text{if } \mathsf{In}(e) = \mathsf{In}(e') \text{ and } h(e) = h(e') \text{ then } e = e' \quad .$$

The branching process (S,h) is called *safe* if S is safe. We note the following technical property of safe branching processes.

Lemma 6. *Let (S,h) be a safe branching process of a net N. Then no spontaneous non-isolated transition of N can occur in the range of h.*

Proof. Let t be a spontaneous and non-isolated transition of N, and suppose that $t = h(e)$ for some event e of S. Then $\mathsf{In}(e) = 0$ and $\mathsf{Out}(e) \neq 0$ by Lemma 3, so e is spontaneous and not isolated, which implies immediately that S is not safe. Thus t is not in the range of h. □

We now introduce a new property of branching processes. We say that the branching process (S,h) is *true* if h is one-to-one on $\overline{q_0}$, the support of the initial marking of S, and on the post-set $e^\bullet$ of each event e of S. Our first observation states that for safe nets, all branching processes are both safe and true.

Lemma 7. *Every branching process of a weakly safe net is safe and true.*

Proof. Let N be a weakly safe net and let (S,h) be a branching process of N. By Lemma 3, we know that S is weakly safe and (S,h) is true. Since S is an occurrence net, it follows from Proposition 3 that S is safe, which concludes the proof. □

Let (S,h) and (S',h') be branching processes of N and let $g: S \to S'$ be a homomorphism of nets. We say that g is a *homomorphism of branching processes* — written $g: (S,h) \to (S',h')$ — if $h = h' \circ g$.

Lemma 8. *Let S and S' be occurrence nets, $h: S \to N$, $h': S' \to N$ and $g: S \to S'$ be homomorphisms such that $h = h' \circ g$ and assume that (S',h') is a branching process of N.*

1. *(S,h) is a branching process of N if and only if (S,g) is a branching process of S'.*
2. *If (S',h') is true, then (S,h) is a true branching process of N if and only if (S,g) is a true branching process of S'.*
3. *If (S',h') is true, then g is the only net homomorphism from S to S'.*
4. *If (S,h) and (S',h') are true branching processes, then g is injective.*

Proof. Let $S=(B,E,\mathsf{In},\mathsf{Out},q_0)$, $S'=(B',E',\mathsf{In}',\mathsf{Out}',q_0')$ and let e,e' be events of S such that $\mathsf{In}(e)=\mathsf{In}(e')$. We first assume that (S,h) is a branching process of N and that $g(e)=g(e')$. Then $h(e)=h'(g(e))$ and $h(e')=h'(g(e'))$ are equal: as (S,h) is a branching process, it follows that $e=e'$. Thus (S,g) is a branching process.

Conversely, suppose that (S,g) is a branching process and $h(e)=h(e')$. Since g is a homomorphism, we have $\mathsf{In}'(g(e))=g(\mathsf{In}(e))$ and $\mathsf{In}'(g(e'))=g(\mathsf{In}(e'))$, so that $\mathsf{In}'(g(e))=\mathsf{In}'(g(e'))$. Moreover, $h'(g(e))=h(e)=h(e')=h'(g(e'))$: since (S',h') is a branching process, we have $g(e)=g(e')$, and since (S,g) is a branching process as well, we have $e=e'$, which concludes the proof of Property (1).

The verification of the preservation of true branching processes, that is, of Property (2), is elementary.

We now turn to Property (3): let us assume that (S',h') is true and that $g'\colon S\to S'$ is a homomorphism such that $h=h'\circ g'$. Let $b\in\overline{q_0}$: then $h(b)=h'(g(b))=h'(g'(b))$ with $g(b),g'(b)\in\overline{q_0'}$. Since h' is injective on $\overline{q_0'}$, it follows that $g(b)=g'(b)$. Now suppose that I is a subnet of S and g and g' coincide on I. Let e be an event of S such that ${}^\bullet e\subseteq I$. Then $\mathsf{In}'(g(e))=g(\mathsf{In}(e))=g'(\mathsf{In}(e))=\mathsf{In}'(g'(e))$. In addition, we have $h'(g(e))=h(e)=h'(g'(e))$: it follows that $g(e)=g'(e)$ since (S',h') is a branching process. Moreover, if $b\in e^\bullet$, then $h'(g(b))=h(b)=h'(g'(b))$, and since h' is injective on $g(e)^\bullet$, we have $g(b)=g'(b)$. Thus $I\cup\{e\}\cup e^\bullet$ is a subnet of S on which g and g' coincide. The hypothesis of quasi-liveness made on (S,h) now suffices to conclude that $g=g'$.

Finally, we verify Property (4): let us assume that both (S,h) and (S',h') are true branching processes. Let $b_1\in\overline{q_0}$ and let b_2 be a condition of S such that $b'=g(b_1)=g(b_2)$. Then $b'\in\overline{q_0'}$. It follows that $b_2\in\overline{q_0}$: if it is not the case, then S has an event $e={}^\bullet b_2$ and $b'=g(b_2)\in g(e)^\bullet$, a contradiction. By Property (2), (S,g) is true, so g is injective on $\overline{q_0}$ and it is now immediate that $b_1=b_2$.

Let I be a subnet of S such that $g^{-1}(g(I))=I$ and g is injective on I. Let e_1,e_2 be events of S such that ${}^\bullet e_1\subseteq I$ and $g(e_1)=g(e_2)=e'$. Since $\mathsf{In}'(e')=g(\mathsf{In}(e_1))=g(\mathsf{In}(e_2))$, the hypothesis on I implies that ${}^\bullet e_2\subseteq I$, and $\mathsf{In}(e_1)=\mathsf{In}(e_2)$. Since (S,g) is a true branching process of S' (by Property (2)), it follows that $e_1=e_2$.

Now if b_1,b_2 are conditions of S such that $b_1\in e^\bullet$ and $g(b_1)=g(b_2)=b'$, then $b_2\notin\overline{q_0'}$ (by the argument developed above). In particular, we have $g(e_1)={}^\bullet b'=g({}^\bullet b_2)$ and, by hypothesis on I, we find that ${}^\bullet b_2\in I$ and $e_1={}^\bullet b_2$. But g is injective on ${e_1}^\bullet$, which contains both b_1 and b_2, so $b_1=b_2$. As in the proof of Property (3), using the quasi-liveness of S, we conclude that g is an injective homomorphism. □

4.2 Unfoldings of a Net

Let (S,h) and (S',h') be branching processes of a net N. As in [2], we say that $(S,h)\sqsubseteq(S',h')$ if S is a subnet of S' (that is, $S\sqsubseteq S'$) and h is the restriction of h' to S. Equivalently,

$(S,h) \sqsubseteq (S',h')$ if there exists an injective homomorphism of branching processes from (S,h) into (S',h').

Now, a $\sqsubseteq$-maximal branching process of N is called an *unfolding* of N.

We say that a marking v of a net is *covered* if there exists a reachable marking m such that $v \leq m$. The following characterization of unfoldings plays a major rôle in the sequel.

Proposition 6. *A branching process (S,h) of a net N is an unfolding if and only if the following property holds:*

(†) if t is a transition of N such that $\mathsf{Pre}(t) = h(v)$ for some marking v covered in S, then there exists an event e of S such that $h(e) = t$ and $\mathsf{In}(e) = v$.

Proof. Let us first assume that (S,h) and (S',h') are branching processes of N, (S,h) satisfies Property (†), and $(S,h) \sqsubseteq (S',h')$. We first verify that if S and S' contain the same events, then they also contain the same conditions, and hence they are equal. Indeed, the assumption that $S \sqsubseteq S'$ implies that S and S' have the same initial conditions and that, for each event e of S', S contains all the conditions of S' which are in ${}^\bullet e$ or $e^\bullet$; finally the definition of occurrence nets implies that all the conditions of S' are either initial or in the postset of an event.

Thus if $S \neq S'$, there is an event e' of S' not in S. In the acyclic net S', we can choose e' to be $\leq$-minimal.

Since S' is quasi-live, there exists a firable sequence of events σ such that the sequence $\sigma e'$ is firable; in addition, σ can be assumed to consist only of events that are $\leq$-less than e'. By minimality of e', σ is also a firable sequence of S and in particular, the conditions in $\mathsf{In}'(e')$ are in S. If v is the marking reached after the firing of σ in S', $m'_0\,[\sigma\rangle\, v$, then the support of v is in S by definition of full subnets, so that $m_0\,[\sigma\rangle\, v$ in S as well.

Now, $\mathsf{Pre}(h'(e')) = h'(\mathsf{In}'(e'))$ and $\mathsf{In}'(e')$ is covered by the marking v, which is reachable in S. By Property (†), there exists an event e of S such that $h(e) = h'(e')$ and $\mathsf{In}(e) = \mathsf{In}'(e')$. But h is a restriction of h', so $h'(e) = h'(e')$ and, by definition of a branching process, we get $e = e'$, a contradiction. Thus, if (S,h) satisfies Property (†), then (S,h) is $\sqsubseteq$-maximal among the branching processes of N, that is, (S,h) is an unfolding of N.

Conversely, let us assume that (S,h) does not satisfy Property (†), that is, there exists a transition t of N which does not lie in $h(S)$, such that $\mathsf{Pre}(t) = h(v)$ for some marking v of S, which is covered in S. Then there exists a firable sequence of S, σ, such that $v \leq \mathsf{Cut}(\sigma)$. Let us construct a new net S' by adding to S one new event, say e, and a set Q of new conditions, equipped with a bijection θ onto the support of $\mathsf{Post}(t)$. We let $\mathsf{Out}'(e) = \sum_{q\in Q} \mathsf{Post}(\theta(q),t)q$, $\mathsf{In}'(e) = v$, and for each event f of S, $\mathsf{In}'(f) = \mathsf{In}(f)$ and $\mathsf{Out}'(f) = \mathsf{Out}(f)$. We also consider the extension h' of the homomorphism h given by $h'(e) = t$ and $h'(q) = \theta(q)$ for each $q \in Q$. In order to conclude, we need to verify that (S',h') is a branching process of N and $(S,h) \sqsubseteq (S',h')$, which is immediate. This shows that (S,h) is not $\sqsubseteq$-maximal, and hence not an unfolding. □

The proof of Proposition 6 yields the following corollary.

Corollary 3. *A true (resp. safe) branching process of a net N is an unfolding if and only if it is $\sqsubseteq$-maximal among the true (resp. safe) branching processes of N.*

Proof. If the true (resp. safe) branching process (S,h) is an unfolding of N, then it is obviously $\sqsubseteq$-maximal among the true (resp. safe) branching processes of N. To verify the converse, suppose that (S,h) is true (resp. safe) and enjoys that relative maximality condition. If (S,h) is not an unfolding, we can construct a branching process (S',h') as in the last paragraph of the proof of Proposition 6. But it is easily verified that (S',h') is true if (S,h) is. It is also clear that S' is weakly safe if S is, and in view of Proposition 3, this means that S' is safe if S is. Thus, if (S,h) is maximal relative to the true (resp. safe) branching processes of N, then it is an unfolding. □

We also note the following corollary of Proposition 6, which expresses the fact that unfoldings simulate all the firable sequences of a net.

Corollary 4. *Let (S,h) be an unfolding of the net N. If σ is a firable sequence of N, then there exists a firable sequence ρ of S such that $h(\rho) = \sigma$.*

Proof. Let $N = (P,T,\mathsf{Pre},\mathsf{Post},m_0)$ and $S = (B,E,\mathsf{In},\mathsf{Out},q_0)$. We first observe the following elementary fact: let m be a multiset over P and v be a multiset over B such that $m \leq h(v)$. Then there exists a multiset v' over B such that $m = h(v')$ and $v' \leq v$. Indeed, for each $p \in P$, we have $m(p) \leq h(v)(p) = \sum_{b\in B,\ h(b)=p} v(b)$: one can choose, for each $b \in B$ such that $h(b) = p$, a value $0 \leq v'(b) \leq v(p)$ such that $m(p) = \sum_{b\in B,\ h(b)=p} v'(b)$.

We now proceed by induction on the length n of σ. If $n = 0$, the statement is trivially true. We now assume that $\sigma = \sigma' t$ and we let w be the marking such that $m_0\ [\sigma'\rangle\ w$. Since σ is firable, $\mathsf{Pre}(t) \leq w$. And by induction hypothesis, there exists a firable sequence ρ' of S such that $h(\rho') = \sigma'$. In particular, if $q_0\ [\rho'\rangle\ v$, we have $w = h(v)$ and hence $\mathsf{Pre}(t) \leq h(v)$. As verified above, it follows that $\mathsf{Pre}(t) = h(v')$ for some multiset $v' \leq v$. In particular, v' is covered in S and by Proposition 6, there exists an event e of S such that $h(e) = t$ and $\mathsf{In}(e) = v'$. Therefore, the event e is enabled by the marking v and hence the sequence $\rho = \rho' e$ is firable, which concludes the proof. □

4.3 True Unfolding and Safe Unfolding

The main results of this section, Theorems 1 and 2, show the existence and the unicity of a true (resp. safe) unfoldings – up to isomorphism in the case of safe unfoldings.

The True Case

Theorem 1. *Every net N has a greatest true branching process, denoted by $\mathcal{B}_{\mathsf{true}}$. Moreover, $\mathcal{B}_{\mathsf{true}}$ is the unique (up to isomorphism) true unfolding of N.*

Proof. Let $N = (P,T,\mathsf{Pre},\mathsf{Post},m_0)$. We first consider the net S_0, whose condition set is $\overline{m_0}$, with initial marking m_0, and without any event. Then S_0 is an occurrence net and if h_0 is the identity map on $\overline{m_0}$, then (S_0,h_0) is a true branching process of N.

Suppose that we have constructed a sequence $(S_i,h_i)_{i\leq n}$ of true branching processes, such that $(S_i,h_i) \sqsubseteq (S_j,h_j)$ whenever $i < j$. By definition, S_i is a subnet (and a prefix) of S_j and h_i is the restriction to S_i of h_j. A new net S_{n+1} is constructed from S_n as follows.

For each transition t of N and for each marking v of S_n, covered in S_n, such that $\mathsf{Pre}(t) = h_n(v)$, and such that S_n has no event e with $h_n(e) = t$ and $\mathsf{In}_n(e) = v$, we add

to S_n a new event e, and a set of conditions Q, equipped with a bijection θ onto the support of $\mathsf{Post}(t)$ (e and the set Q depend on the choice of t and v). Then we extend the mappings In_n and Out_n by letting $\mathsf{In}_{n+1}(e) = v$ and $\mathsf{Out}_{n+1}(e) = \sum_{q\in Q} \mathsf{Post}(\theta(q),t)q$ for each new event. We also extend h_n by letting $h_{n+1}(e) = t$ for each new event e, and $h_{n+1}(q) = \theta(q)$ for each new condition q. It is immediate that the resulting net S_{n+1} is an occurrence net, that the pair (S_{n+1}, h_{n+1}) is a true branching process of N, and that $(S_n, h_n) \sqsubseteq (S_{n+1}, h_{n+1})$.

Note that S_{n+1} may not be defined (if no pair (t,v) as above can be identified in S_n), in which case the sequence $(S_i, h_i)_i$ is finite. In general however, this is an infinite increasing sequence of true branching processes of N. In any case, let $S = (B, E, \mathsf{In}, \mathsf{Out}, q_0)$ be the union of the S_i, that is, $B = \bigcup_{i\in I} B_i$ and $E = \bigcup_{i\in I} E_i$, with $\mathsf{In}(e) = \mathsf{In}_i(e)$ and $\mathsf{Out}(e) = \mathsf{Out}_i(e)$ if $e \in E_i$. The mappings $h\colon B \to P$ and $h\colon E \to T$ are defined similarly, by letting $h(x) = h_i(x)$ whenever x occurs in S_i. Again, it is easily verified that (S,h) is a true branching process of N. We now verify that (S,h) is an unfolding, using Condition (†) in Proposition 6.

Let t be a transition of N such that $\mathsf{Pre}(t) = h(v)$ for some marking v covered in S. Then t lies in some S_n, and since each S_n is a prefix of S, the marking v is also covered in S_n and $\mathsf{Pre}(t) = h_n(v)$. By construction, there exists an event e in S_n or in S_{n+1} such that $h_{n+1}(e) = t$ and $\mathsf{In}_{n+1}(e) = v$. In particular $h(e) = t$ and $\mathsf{In}(e) = v$, so Condition (†) holds and (S,h) is an unfolding.

To prove uniqueness of the true unfolding, we consider true unfoldings (S,h) and (S',h'). By Proposition 6, one can verify that h and h' are surjective onto $\overline{m_0}$, and hence they establish bijections from $\overline{q_0}$ and $\overline{q'_0}$ onto $\overline{m}_0$. For each condition b in $\overline{q_0}$, we let $g(b)$ be the unique condition in $\overline{q'_0}$ such that $h(b) = h'(g(b))$. Suppose now that a homomorphism g has been defined from a prefix of (S,h) to (S',h'), and let e be a $\leq$-minimal event of S which is not in the domain of g. Since S is quasi-live, there exists a firable sequence of events σe, where σ consists only of events that are $\leq$-less than e, and hence are in the domain of g. In particular, $\mathsf{In}(e)$ is covered by $\mathsf{Cut}(\sigma)$, and the conditions in the support of $\mathsf{In}(e)$ are in the domain of g. Consider then the transition $h(e)$ in N. Then $\mathsf{Pre}(h(e)) = h(\mathsf{In}(e)) = h'(g(\mathsf{In}(e)))$ and this marking is covered in S' by $g(\sigma)$. By Condition (†), there exists an event e' of S' such that $h'(e') = h(e)$ and $\mathsf{In}'(e') = g(\mathsf{In}(e))$. Moreover, by definition of true branching processes, h and h' define bijections from $e^\bullet$ and $e'^\bullet$ to $h(e)^\bullet$. Thus, if we add to the domain of g the event e and the conditions in its postset $e^\bullet$, we can extend g to a homomorphism defined on this larger prefix of S.

Since every condition and event of S has a finite past (Proposition 2) and since such homomorphisms defined on prefixes of S must coincide on the intersection of their domains (Lemma 8), we can define a homomorphism from the branching process (S,h) to the branching process (S',h').

By Lemma 8 again, this morphism is injective, so $(S,h) \sqsubseteq (S',h')$ and the $\sqsubseteq$-maximality of unfoldings implies that $(S,h) = (S',h')$. □

We record the following easy but illuminating observation.

Corollary 5. *Each true branching process of a net N is isomorphic to a single prefix of the true unfolding of N.*

Proof. Since $\mathcal{B}_{\mathsf{true}}$ is the unique true unfolding of N (Theorem 1), every true branching process (S,h) of N is $\sqsubseteq$-less than $\mathcal{B}_{\mathsf{true}}$, and hence there exists an injective homomorphism $g\colon (S,h) \to \mathcal{B}_{\mathsf{true}}$. In particular, (S,h) is isomorphic to its image under g, which is a prefix of $\mathcal{B}_{\mathsf{true}}$. Moreover, Lemma 8 shows that this homomorphism is unique, and hence (S,h) is not isomorphic to any other prefix of $\mathcal{B}_{\mathsf{true}}$. □

The existence of a greatest true branching process can also be used to show the following, more detailed order-theoretic result.

Proposition 7. *The set of true branching processes forms a complete lattice.*

Proof. Let $\mathcal{B}_{\mathsf{true}}$ be the unique true unfolding of N, and let $(S_i,h_i)_{i\in I}$ be a family of true branching processes of N. Then each S_i can be identified with a subnet of $\mathcal{B}_{\mathsf{true}}$. The proof of the announced result thus reduces to verifying that an arbitrary intersection or union of sub-occurrence nets of an occurrence net form an occurrence net.

But Lemma 5 shows that a subnet of $\mathcal{B}_{\mathsf{true}}$ is an occurrence net itself if and only if it is a prefix – and it is immediate that an intersection or a union of prefixes is a prefix. □

The Safe Case. With a similar proof as Theorem 1, we get an analogous result, concerning the existence of a safe unfolding. However, as we shall see, the order-theoretic properties of safe branching processes are less strong than for true branching processes.

Theorem 2. *Suppose that every spontaneous transition of N is isolated. Then N has a greatest safe branching process, denoted by $\mathcal{B}_{\mathsf{safe}}$. Moreover, up to isomorphism, $\mathcal{B}_{\mathsf{safe}}$ is the unique safe unfolding of N.*

The proof follows the same line as that of Theorem 1 and is given in appendix.

Example 1. Consider the net N represented in Figure 1. The nets $\beta_1,\ldots,\beta_4$ represented in Figure 2 are safe branching processes of N. Moreover, it is easy to verify that both β_3 and β_4 are $\sqsubseteq$-less than β_1 and β_2, and that both are maximal with this property: thus β_1 and β_2 do not admit a greatest lower bound. Similarly, Figure 3 shows safe branching processes β_1 and β_2, and distinct branching processes β_3 and β_4 which are minimal among the common upper bounds of β_1 and β_2. Thus β_1 and β_2 do not admit a least upper bound.

It is interesting to observe (compare with Corollary 5) that in both cases, both β_1 and β_2 are isomorphic to two distinct subnets of the safe unfolding of N, represented in Figure 4.

5 Processes and their Properties

The notion of process of a net is developped in the literature [7,2], but it is limited to the case of safe processes. Here we extend the notion, in such a way that the usual *processes* are what we call here *safe processes*.

By definition, a *process of a net N* is a triple $\pi = (S,h,\varphi)$ such that (S,h) is a branching process of N, and φ is a configuration of S which covers all its events (that is, every event of S is in the support of φ).

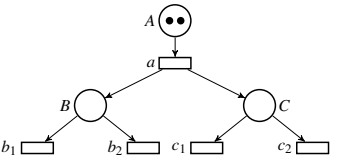


Fig. 1. The net N

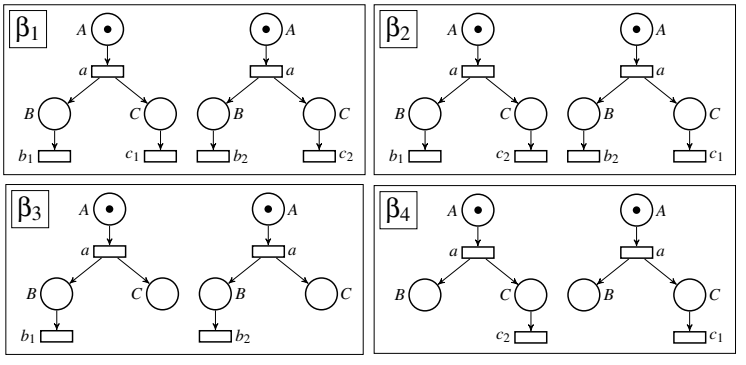


Fig. 2. Safe branching processes may not have an inf

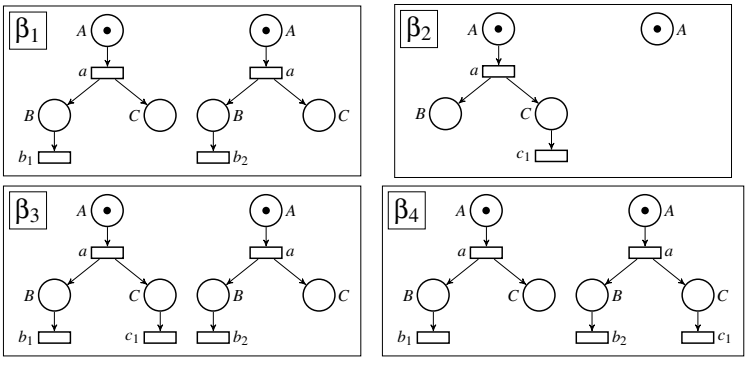


Fig. 3. Safe branching processes may not have a sup

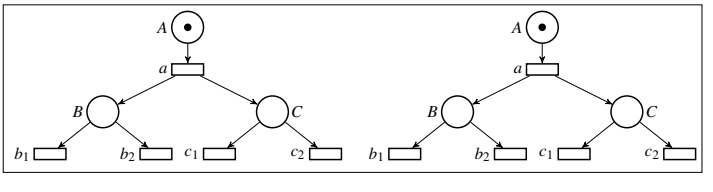


Fig. 4. The safe unfolding of N

We also extend the partial order on branching processes of N, to its processes. If $\pi_1 = (S_1, h_1, \varphi_1)$ and $\pi_2 = (S_2, h_2, \varphi_2)$ are processes of a net N, we say that π_1 and π_2 are *equivalent*, written $\pi_1 \equiv \pi_2$, if there exists an isomorphism of branching processes g, from (S_1, h_1) to (S_2, h_2), with $g(\varphi_1) = \varphi_2$. We say that π_1 is *smaller than* π_2, written $\pi_1 \sqsubseteq \pi_2$, if there exists an injective homomorphism from (S_1, h_1) to (S_2, h_2) with $g(\varphi_1) \leq \varphi_2$. It is immediate that $\sqsubseteq$ defines a partial order among equivalence classes of processes of N.

Finally, we say that a process $\pi = (S, h, \varphi)$ is *true* (resp. *safe*) if the underlying branching process (S, h) is true (resp. safe). Moreover, if (S', h') is a branching process of N and $(S, h) \sqsubseteq (S', h')$, we say that π is a *process of the branching process* (S', h').

5.1 True Processes

The above definitions lead directly to the following lemma.

Lemma 9. *Every true (resp. safe) process of a net N is a process of the true (resp. safe) unfolding of N.*

Proof. The statement follows from Theorems 1 and 2, since up to isomorphism, every true (resp. safe) branching process of N is a prefix of its true (resp. safe) unfolding. □

Lemma 9 then leads to the following characterization of the true unfolding of a net.

Proposition 8. *If a branching process of N contains all the true processes of N, then it is the true unfolding of N.*

Proof. Let (S, h) be a branching process containing all the true processes of N. We first assume that (S, h) is true. Then the underlying branching processes of the true processes of N occur as prefixes of (S, h), and by Lemma 8, each one occurs in a unique way.

Now consider an event e of $\mathcal{B}_{\mathsf{true}}$. The quasi-liveness of occurrence nets ensures that e occurs in a process π of the branching process $\mathcal{B}_{\mathsf{true}}$, and π is necessarily true. Therefore π is a branching process of (S, h). Since this holds for each event e, it follows that $\mathcal{B}_{\mathsf{true}} \sqsubseteq (S, h)$. The $\sqsubseteq$-maximality of $\mathcal{B}_{\mathsf{true}}$ allows us to conclude that (S, h) is the true unfolding of N.

In the general case, where (S, h) is not assumed to be true, let R be the subnet of S consisting of the initial conditions, of the events e such that, for each event $f \leq e$, h is injective on $f^\bullet$, and of the pre- and post-conditions of these events. It is immediate that R is a prefix of S and then an occurrence net (Lemme 5). If k is the restriction of h to R then $(R, k) \sqsubseteq (S, h)$. Moreover, (R, k) is a true branching process, and every true process of (S, h) is in fact a true process of (R, k). It follows from the first part of the proof that $(R, k) = \mathcal{B}_{\mathsf{true}}$, and by the $\sqsubseteq$-maximality of $\mathcal{B}_{\mathsf{true}}$, that $(S, h) = \mathcal{B}_{\mathsf{true}}$. □

We can also show that true processes have valuable order-theoretic properties with respect to the $\sqsubseteq$-order.

Proposition 9. *Any family of true processes of a net N admits a greatest lower bound. If a family of true processes of N admits a common upper bound, then it has a least upper bound.*

Proof. Let $(\pi_i)_{i\in I}$ be a family of true processes of N, with $\pi_i = (S_i, h_i, \varphi_i)$ for each i. Each (S_i, h_i) can be viewed in an unambiguous fashion as a prefix of $\mathcal{B}_{\mathsf{true}}$ (Corollary 5), and hence φ_i can be viewed as a configuration of $\mathcal{B}_{\mathsf{true}}$. Let $\varphi = \min_i \varphi_i$, let S be the subnet of $\mathcal{B}_{\mathsf{true}}$ consisting of the initial conditions, the events occurring in φ and their pre- and post-conditions. It is easily verified that S is a prefix of $\mathcal{B}_{\mathsf{true}}$, and if h is the restriction of (any) h_i to S, (S,h) is a true branching process of N. It is also clear that $(S,h) \sqsubseteq (S_i, h_i)$ and $\varphi \leq \varphi_i$ for each i. We want to show that $(S,h,\varphi) = \min_i \pi_i$, and for this purpose, it suffices to establish that φ is a configuration. We use the characterization given in Corollary 2.

First it is immediate that φ has finite support, since each φ_i does. Moreover, the fact that each φ_i satisfies the inequalities in Corollary 2 easily implies that φ does as well. Thus $(S,h,\varphi) = \min_i \pi_i$.

The statement concerning upper bounds follows immediately: if the set U of upper bounds of $(\pi_i)_{i\in I}$ is non-empty, we claim that $\inf U$ is the least upper bound of the $(\pi_i)_{i\in I}$. To justify this claim, it suffices to verify that $\inf U$ is indeed an upper bound, that is, $\inf U \in U$. This is readily verified, using the description of $\inf U$ in the first part of the proof, and the fact that the configurations we consider are finite support vectors with positive integer coefficients. □

Remark 2. Let $(\pi_i)_{i\in I}$ be a family of true processes of a net N as in Proposition 9. It is not difficult to verify that this family does not have an upper bound if it takes infinitely many values. Now consider the net N in Figure 1, and consider the true processes determined by the configurations $a+b_1$, $a+b_2$ and $a+c_1$. Any two of these processes have a least upper bound: for instance $\sup(a+b_1, a+b_2) = 2a+b_1+b_2$. Note in particular that the sup of these configurations taken as multisets, $a+b_1+b_2$, is not a configuration. Moreover, the three processes taken together do not admit a common upper bound.

5.2 Safe Processes

In this section, we verify that safe processes do not have the good order-theoretic properties enjoyed by safe processes, described in Propositions 8 and 9.

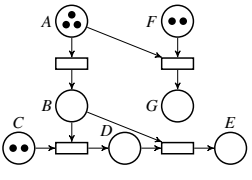


Fig. 5. The net N

Consider the net N represented in Figure 5. Figure 6 shows all the $\sqsubseteq$-maximal safe processes of N: there are 4 of them, and in particular, no two of these processes admit a sup.

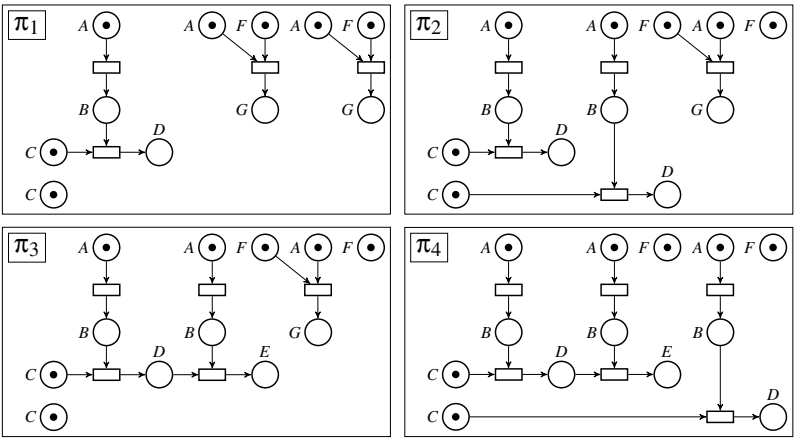


Fig. 6. All the maximal safe processes of N

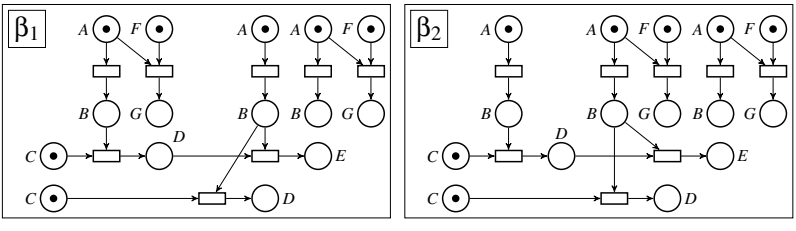


Fig. 7. Two safe branching processes containing all the safe processes of N

Moreover, Figure 7 shows two distinct safe branching processes of N, both of which contain all the maximal safe processes. Thus Proposition 8 does not hold for safe processes.

Of course, the safe branching processes of Figure 7 also occur in the safe unfolding of N. This safe unfoldings presents large-scale duplication of safe processes.

6 Concluding Remarks

This paper proposes a general framework for the unfolding of general Petri nets. Traditional occurrence nets are covered by safe unfoldings. The drawbacks of this approach, when applied to general Petri nets, were identified by Hoogers *et al.* [4], and examples are given in this paper as well, in Sections 4.3 and 5.2: even though every net admits a unique safe unfolding, the safe branching processes do not form a lattice, and neither do the safe processes. The root of the problem can be traced to the following fact: a safe process of a net N may occur in several ways as a process of the safe unfolding of N.

The essential contribution of our work is the concept of true unfolding of a net. This is an extension of the traditional notion in the following sense: if N is a safe Petri net, then its safe and its true unfoldings (resp. branching processes, processes) coincide. The true processes and the true branching processes of a general net satisfy good order-theoretic properties. Moreover, true branching processes as well as true processes have a unique representation in the true unfolding.

One question not tackled in this paper is the relationship between unfoldings and event structures. Within the framework of general Petri nets, the concept of prime event structure is not adapted to capture the multiset aspects. Two natural questions should be considered: (1) In which cases does the notion of true unfolding make it possible to capture the conflict and causality relations? (2) What is the concept of event structures associated to true unfoldings? An additionnal question concerns the interpretation of true unfoldings to define a true concurrency semantics similarly to [8].

References

1. Best, E., Devillers, R.: Sequential and concurrent behaviour in Petri net theory. Theoretical Computer Science 55, 87–136 (1987)
2. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica 28, 575–591 (1991)
3. Haar, S.: Branching processes of general S/T-systems and their properties. Electronic Notes in Theoretical Computer Science 18 (1998)
4. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theoretical Computer Science 153, 129–170 (1996)
5. 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)
6. Murata, T.: Petri nets: Properties, analysis, and applications. Proc. of the IEEE 77(4), 541–580 (1989)
7. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, events structures and domains, Part I. Theoretical Computer Science 13(1), 85–108 (1981)
8. van Glabbeek, R., Plotkin, G.: Configuration structures, event structures and Petri nets. Theoretical Computer Science 410(41), 4111–4159 (2009)
9. Vogler, W.: Executions: A new partial-order semantics of Petri nets. Theoretical Computer Science 91, 205–238 (1991)

Refinement of Synchronizable Places with Multi-workflow Nets

Weak Termination Preserved!

Kees M. van Hee, Natalia Sidorova, and Jan Martijn van der Werf*

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee,n.sidorova,j.m.e.m.v.d.werf}@tue.nl

Abstract. Stepwise refinement is a well-known strategy in system modeling. The refinement rules should preserve essential behavioral properties, such as deadlock freedom, boundedness and weak termination. A well-known example is the refinement rule that replaces a safe place of a Petri net with a sound workflow net. In this case a token on the refined place undergoes a procedure that is modeled in detail by the refining workflow net.

We generalize this rule to component-based systems, where in the first, high-level, refinement iterations we often encounter in different components places that represent in fact the counterparts of the same procedure "simultaneously" executed by the components. The procedure involves communication between these components.

We model such a procedure as a multi-workflow net, which is actually a composition of communicating workflows. Behaviorally correct multi-workflow nets have the weak termination property. The weak termination requirement is also applied to the system being refined. We want to refine selected places in different components with a multi-workflow net in such a way that the weak termination property is preserved through refinements. We introduce the notion of synchronizable places and show that a sufficient condition for preserving weak termination is that the places to be refined are synchronizable. We give a method to decide if a given set of places is synchronizable.

1 Introduction

Complex systems are often build from components, each component having its own dedicated set of functionality. At runtime, components communicate with each other to accomplish their tasks. Every separate component can still be very complex. Therefore, an important principle in modeling component-based systems is *refinement*. In several iterations a model is refined from an abstract

* Supported by the PoSecCo project (project no. 257129), partially co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7)

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 149–168, 2011.

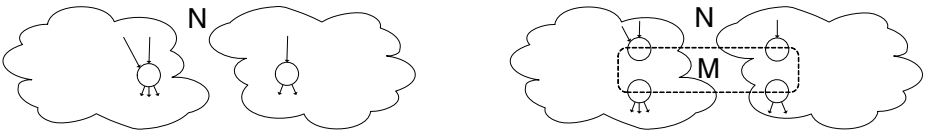


Fig. 1. Refinement of (synchronizable) places

model to a more precise model. When modeling the behavior of a system using Petri nets, one can, e.g., represent some procedure by a place in a Petri net, assuming that a token undergoes this procedure when being in this place, and later on, refine this place with the actual procedure, modelled by a workflow net.

When working with *component-based systems*, different components can contain places that together represent a single procedure. To refine the system with the actual procedure, we need to refine these places simultaneously by the procedure, as shown in Fig. 1. The same scheme can be used in the context of *Service-Oriented architectures* (SOA), where communicating services might make use of other services. The model of the procedure is then a composition of communicating workflow nets modeling the component procedures.

Consider a simple example of a procedure for booking a trip, with the system divided into three components: a travel agency, an airline and a hotel chain. The component of the agency contains place "booking a trip". In the component of the airline there is a place called "booking a seat", and in the hotel component some place "booking a room" exists. These places are related by an underlying booking procedure. In the procedure, a seat is selected at the airline, which may involve several cycles and communication with the client at the agency. Next, a reservation is made for a hotel room, which again may involve several iterations. Finally, the agency confirms the reservation at the airline. When refining the system design, we would like to refine these three places by three communicating workflow nets.

To model such a partitioned procedure, we introduce *multi-workflow nets*, being a generalization of workflow nets. Then we define the *refinement of a set of places by a multi-workflow net*, which is a generalization of the place refinement with a workflow net from [8]. A natural question that arises then is *under which conditions properties of interest are preserved through refinements.*

The property we focus on in this paper is *weak termination*, meaning that from every reachable state of a system some final state can be reached. Given a weakly terminating system with a set of places to be refined and a weakly terminating multi-workflow, we want to guarantee that the refined system is weakly terminating as well. By means of examples we motivate the requirements of "synchronizability" for the set of places to be refined, formalize this requirement and prove that if the requirement holds, *refinement of synchronizable places preserves weak termination.*

The paper is organizes as follows. In Sec. 2 we introduce basic concepts. In Sec. 3 we define the notion of multi-workflow nets and the refinement of a set of places with a multi-workflow. In Sec. 4 we give the intuition for the notion of

synchronizable places and in Sec. 5 we formalize this notion. In Sec. 6 we prove that weak termination is preserved through refinements of sets of synchronizable places. In Sec. 7 we discuss the place of our work among related works and in Sec. 8 we draw conclusions and discuss directions for future work.

2 Preliminaries

Let S be a set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. We use $|S|$ for the number of elements in S. Two sets U and V are *disjoint* if $U \cap V = \emptyset$. We denote the cartesian product of two sets S and T by $S \times T$. On a cartesian product we define two projection functions $\pi_1 : S \times T \to S$ and $\pi_2 : S \times T \to T$ such that $\pi_1((s,t)) = s$ and $\pi_2((s,t)) = t$ for all $(s,t) \in S \times T$. We lift the projection function to sets in the standard way, i.e. $\pi_i(U) = \{\pi_i((s,t) \mid (s,t) \in U\}$ for $U \subseteq A \times B$ and $i \in \{1,2\}$.

A *bag* m over S is a function $m : S \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, \ldots\}$ denotes the set of natural numbers. We denote e.g. the bag m with an element a occurring once, b occurring three times and c occurring twice by $m = [a, b^3, c^2]$. The set of all bags over S is denoted by $\mathbb{N}^S$. Sets can be seen as a special kind of bag where all elements occur only once; we interpret sets in this way whenever we use them in operations on bags. 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. The projection of a bag $m \in \mathbb{N}^S$ on elements of a set $U \subseteq S$, is denoted by $m_{|U}$, and is defined by $m_{|U}(u) = m(u)$ for all $u \in U$ and $m_{|U}(u) = 0$ for all $u \in S \setminus U$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \ldots, n\}$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. 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 all finite sequences over S is denoted by S^*. Let $\nu, \gamma \in S^*$ be two sequences. *Concatenation*, denoted by $\sigma = \nu; \gamma$ is defined as $\sigma : \{1, \ldots, |\nu| + |\gamma|\} \to S$, such that for $1 \leq i \leq |\nu|$: $\sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$: $\sigma(i) = \gamma(i - |\nu|)$. *Projection* of a sequence $\sigma \in S^*$ on elements of a set $U \subseteq S$, denoted by $\sigma_{|U}$, is inductively defined by $\epsilon_{|U} = \epsilon$ and $(\langle a \rangle; \sigma)_{|U} = \langle a \rangle; \sigma_{|U}$ if $a \in U$ and $(\langle a \rangle; \sigma)_{|U} = \sigma_{|U}$ otherwise.

Labeled transition system. A *labeled transition system* (LTS) is a 5-tuple $(S, \mathcal{A}, \longrightarrow, s_0, \Omega)$ where (1) S is a set of *states*; (2) $\mathcal{A}$ is a set of *actions*; (3) $\longrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times S$ is a *transition relation*, where $\tau \notin \mathcal{A}$ is the silent action [1]; (4) $s_0 \in S$ is the *initial state*; and (5) $\Omega \subseteq S$ is the set of *final states*.

Let $L = (S, \mathcal{A}, \longrightarrow, s_0, \Omega)$ be an LTS. For $s, s' \in S$ and $a \in \mathcal{A} \cup \{\tau\}$, we write $(L : s \xrightarrow{a} s')$ iff $(s, a, s') \in \longrightarrow$. If $(L : s \xrightarrow{a} s')$, we say that state s' is *reachable* from s by an action labeled a. A state $s \in S$ is called a *deadlock* if no action $a \in \mathcal{A} \cup \{\tau\}$ and state $s' \in S$ exist such that $(L : s \xrightarrow{a} s')$. We define $\Longrightarrow$ as the smallest relation such that $(L : s \Longrightarrow s')$ if $s = s'$ or $\exists s'' \in S : (L : s \Longrightarrow s'' \xrightarrow{\tau} s')$. As a notational convention, we may write $\xRightarrow{\tau}$ for $\Longrightarrow$. For $a \in \mathcal{A}$ we define $\xRightarrow{a}$ as the smallest relation such that $(L : s \xRightarrow{a} s')$ if $\exists s_1, s_2 \in S : (L : s \Longrightarrow s_1 \xrightarrow{a} s_2 \Longrightarrow s')$. We lift the notations of actions to

sequences. For the empty sequence ϵ, we have $(L : s \xrightarrow{\epsilon} s')$ iff $(L : s \Longrightarrow s')$. A sequence $\sigma \in \mathcal{A}^*$ of length $n > 0$ is a firing sequence from $s_0, s_n \in S$, denoted by $(L : s_0 \xrightarrow{\sigma} s_n)$ if states $s_{i-1}, s_i \in S$ exist such that $(L : s_{i-1} \overset{\sigma(i)}{\Longrightarrow} s_i)$ for all $1 \leq i \leq n$. If a firing sequence σ exists such that $(L : s \xrightarrow{\sigma} s')$ we say that s' is *reachable* from s. The set of all reachable states from s are the states from the set $\mathcal{R}(L, s) = \{s' \mid \exists \sigma \in \mathcal{A}^* : (L : s \xrightarrow{\sigma} s')\}$.

The correctness notion we focus on in this paper is *weak termination*. An LTS $L = (S, \mathcal{A}, \longrightarrow, s_0, \Omega)$ is *weakly terminating* if $\Omega \cap \mathcal{R}(L, s) \neq \emptyset$ for all states $s \in \mathcal{R}(L, s_0)$, i.e. from every state reachable from the initial state some final marking can be reached.

Petri nets. A *Petri net* N is a 3-tuple (P, T, F) where (1) P and T are two disjoint sets of *places* and *transitions* respectively; (2) $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. The elements from the set $P \cup T$ are called the *nodes* of N. Elements of F are called *arcs*. Places are depicted as circles, transitions as squares. For each element $(n_1, n_2) \in F$, an arc is drawn from n_1 to n_2. Two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$ are *disjoint* if and only if $(P \cup T) \cap (P' \cup T') = \emptyset$. Let $N = (P, T, F)$ be a Petri net. Given a node $n \in (P \cup T)$, we define its *preset* ${}^\bullet_N n = \{n' \mid (n', n) \in F\}$, and its *postset* $n^\bullet_N = \{n' \mid (n, n') \in F\}$. We lift the notation of preset and postset to sets and sequences. Given a set $U \subseteq (P \cup T)$, ${}^\bullet_N U = \bigcup_{n \in U} {}^\bullet_N n$ and $U^\bullet_N = \bigcup_{n \in U} n^\bullet_N$. The preset of a sequence $\sigma \in T^*$ is the set of all places that occur in a preset of a transition in σ, i.e., ${}^\bullet_N \sigma = \{p \mid \exists 1 \leq i \leq |\sigma| : p \in {}^\bullet_N \sigma(i)\}$. Likewise, the postset of σ is the set of all places that occur in a postset of a transition in σ, i.e., $\sigma^\bullet_N = \{p \mid \exists 1 \leq i \leq |\sigma| : p \in \sigma(i)^\bullet_N\}$. If the context is clear, we omit the N in the subscript.

Let $N = (P, T, F)$ be a Petri net. A *marking* of N is a bag $m \in \mathbb{N}^P$, where $m(p)$ denotes the number of *tokens* in place $p \in P$. If $m(p) > 0$, place $p \in P$ is called *marked* in marking m. A Petri net N with corresponding marking m is written as (N, m) and is called a *marked Petri net*. A *system* $\mathcal{S}$ is a 3-tuple $((P, T, F), m_0, \Omega)$ where $((P, T, F), m_0)$ is a marked Petri net and $\Omega \subseteq \mathbb{N}^P$ is a set of *final markings*.

The semantics of a system $\mathcal{N} = ((P, T, F), m_0, \Omega)$ is defined by an LTS $\mathcal{S}(\mathcal{N}) = (\mathbb{N}^P, T, \rightarrow, m_0, \Omega)$ where $(m, t, m') \in \longrightarrow$ iff ${}^\bullet t \leq m$ and $m' + {}^\bullet t = m + t^\bullet$ for $m, m' \in \mathbb{N}^P$ and $t \in T$. We write $(N : m \xrightarrow{t} m')$ as a shorthand notation for $(\mathcal{S}(\mathcal{N}) : m \xrightarrow{t} m')$ and $\mathcal{R}(\mathcal{N}, m)$ for $\mathcal{R}(\mathcal{S}(\mathcal{N}), m)$. We say that a place p is *safe*, if $m(p) \leq 1$ for any marking $m \in \mathcal{R}(\mathcal{N}, m_0)$. *Weak termination* of a system corresponds to weak termination of the corresponding transition system.

A *workflow net* W is a 5-tuple (P, T, F, i, f) is a Petri net such that (P, T, F) is a Petri net, $i \in P$ is the initial place and $f \in P$ is the final place such that ${}^\bullet i = f^\bullet = \emptyset$ and all nodes $(P \cup T)$ of N are on a path from i to f. We say that a workflow net is weakly terminating if the system $((P, T, F), [i], [f])$ is weakly terminating.

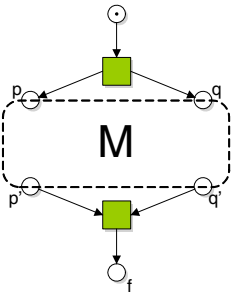
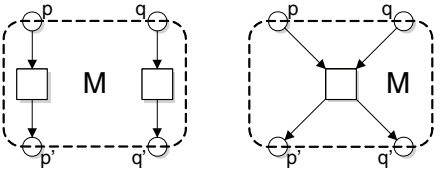


Fig. 2. Soundness skeleton for refining component



(a) Passing (b) Synchronization

Fig. 3. Simple nets to refine with

3 Refinement of Sets of Places

Many refinement/reduction rules exist for Petri nets, like the rules of Murata [13] and Berthelot [2]. Many of those rules guarantee the preservation of weak termination: applying them to a weakly terminating system results again in a weakly terminating system.

Single place refinement. In [8], the authors show that a place in a workflow net may be refined with a generalized sound workflow net, while preserving the weak termination property. In this refinement, any place p can be replaced by a generalized sound workflow net (generalized soundness is weak termination of a workflow for all initial markings $[i^n], n \in I\!N$). All input arcs of p become input arcs of the initial place of the workflow net, and all output arcs of p become output arcs of the final place of the workflow net. For a safe place p, it is enough to require that both the system being refined and the refining workflow are weakly terminating, as proven in Theorem 9 from [8].

Place refinement with a weakly terminating workflow net is very useful in correctness-by-construction approaches based on stepwise refinement (see e.g. [10,14,19]).

When working with component-based information system, we often need more involved refinements: Consider e.g. a component N (modeled as a system) being an asynchronous composition of two components A and B. Suppose that place p in component A and place q in component B model at a high level counterparts of the same procedure in A and B, e.g. the payment procedure. Then p and q get refined by applying some standard workflow subcomponents C and D, possibly communicating to each other. All incoming arcs to place p are connected to the initial place of C, and all outgoing arcs from place p are connected to its final place; likewise for place q. This approach is depicted in Fig. 1; M stands there for the composition of C and D.

Multi-workflow nets. To model a procedure distributed over multiple components (like net M in Fig. 1), we introduce the notion of a *multi-workflow net* (MWF net), which is a generalization of the notion used in [7]. A multi-workflow net has for each component an i/o pair consisting of an input place and an output place. Note that the definition of a MWF net with a single i/o pair coincides with the definition of a classical workflow net where the initial place is the input place and the final place the output place.

Definition 1 (Multi-workflow net). *A* multi-workflow net *(MWF net) N is a 4-tuple (P, T, F, E) where (P, T, F) is a Petri net and $E \subseteq P \times P$ is a set of* i/o pairs*, such that $|E| = |\pi_1(E)| = |\pi_2(E)|$ and ${}^\bullet\pi_1(E) = \pi_2(E)^\bullet = \emptyset$. The places in $\pi_1(E)$ are called the* input places *of N, the places in $\pi_2(E)$ are called the* output places *of N. Furthermore, each node $n \in P \cup T$ is on a path from an input place to an output place.*

The *initial marking* of an MWF net is the marking containing one token on every input place, and the final marking is the marking containing one token on every output place. We enforce the notion of weak termination for multi-workflows with an additional requirement related to the i/o feature of multi-workflows, namely, the two places of every i/o pair should be causally connected, and thus whenever the output place gets marked the corresponding input place does not contain tokens anymore.

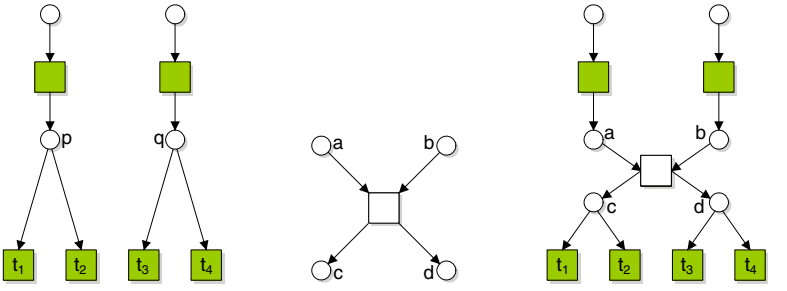
Definition 2 (Weak termination of an MWF net). *An MWF net $N = (P, T, F, E)$ is* weakly terminating *if (1) the system $\mathcal{N} = ((P, T, F), \pi_1(E), \{\pi_2(E)\})$ is weakly terminating and (2) $m(p) + m(q) \leq 1$ for all pairs $(p, q) \in E$ and markings $m \in \mathcal{R}(\mathcal{N}, \pi_1(E))$.*

Similar to the case of classical workflow nets (see Lemma 11 in [9]), it is easy to prove that the only marking reachable in a weakly terminating MWF net that contains the final marking is the final marking itself.

Lemma 3 (Proper completion of an MWF net). *Let $N = (P, T, F, E)$ be a weakly terminating MWF net and $m \in \mathcal{R}(\mathcal{N}, \pi_1(E))$ such that $\pi_2(E) \leq m$. Then $m = \pi_2(E)$.*

Refinement of a set of places. The refinement we are interested in is the refinement of n places of a system $\mathcal{N}$ with an MWF net M. Like in place refinement, each place p belonging to the n selected places is substituted by an i/o pair: the preset of p becomes the preset of the input place of the i/o pair, the postset of p becomes the postset of the output place of the i/o pair. Fig. 4 shows an example of such a refinement. The initial marking of the refined net contains the initial marking of $\mathcal{N}$, with the tokens of the refined places being transferred to the corresponding input places of the MWF net M. Similarly, the set of final markings contains all the final markings of $\mathcal{N}$, with the tokens of the refined places being transferred to the corresponding output places of M.

Definition 4 (Refinement of a set of places). *Let $\mathcal{N} = (N, m_{0_N}, \Omega_N)$ be a system with $N = (P_N, T_N, F_N)$ and $R \subseteq P_N$ be a set of places to be refined and*



(a) Net N (b) Net M (c) Refinement $N \odot M$

Fig. 4. Example of a refinement of a set of places

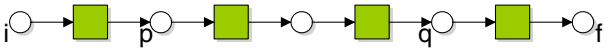


Fig. 5. Linear net N

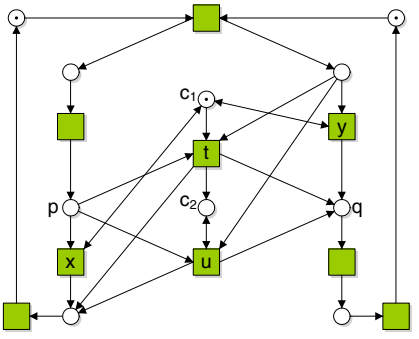
$M = (P_M, T_M, F_M, E_M)$ be a MWF net, such that N and M are disjoint. Let $\alpha : R \to E_M$ be a total, bijective function. The refinement $\mathcal{N} \odot_\alpha M$ is a system $((P, T, F), m_0, \Omega)$ where

- $P = (P_N \setminus R) \cup P_M$;
- $T = T_N \cup T_M$;
- $F = (F_N \setminus \bigcup_{r \in R}(({}_N^\bullet r \times \{r\}) \cup (\{r\} \times r_N^\bullet))) \cup F_M$
  $\cup \bigcup_{r \in R}(({}_N^\bullet r \times \pi_1(\alpha(r))) \cup (\pi_2(\alpha(r)) \times r_N^\bullet))$;
- $m_0 = m_{0_{N|P}} + \sum_{r \in R} m_{0_N}((\pi_1 \circ \alpha)(r))$;
- $\Omega = \{m \mid \exists m_N \in \Omega_N : m = m_{N|P} + \sum_{r \in R} m_N((\pi_2 \circ \alpha)(r))\}$.

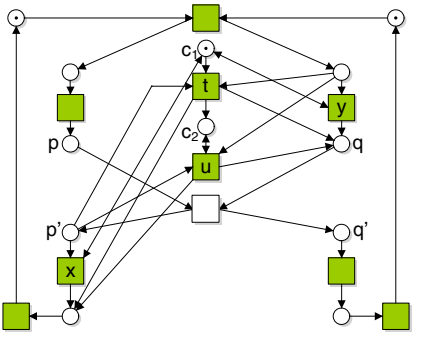
4 Intuition for the Notion of Synchronizable Places

We want to guarantee that the refinement of a set of places in a weakly terminating system by an arbitrary weakly terminating MWF net preserves weak termination. In general, this is not the case. In this section, we consider the refinement of a pair of places, in the next section we generalize the requirements to sets of places.

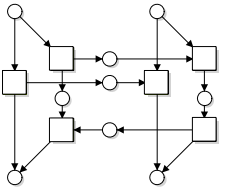
First of all, the *refined places should be safe*, i.e., in any marking reachable in the system, the place is marked with at most one token. This is needed already for the refinements of single places (see [8]). The reason lies in the definition of



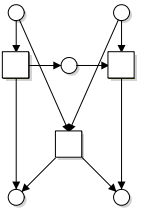
(a) Net N



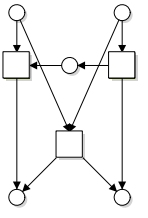
(b) Synchronizing refinement of N



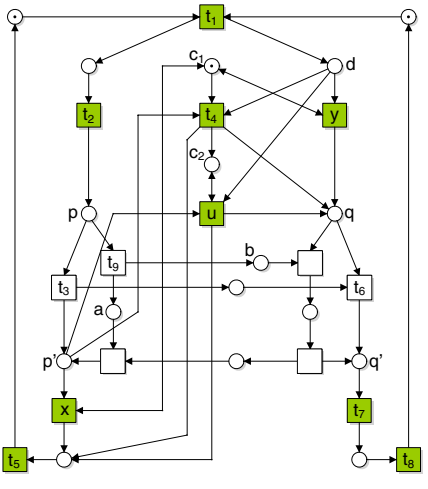
(c) MWF net M_1



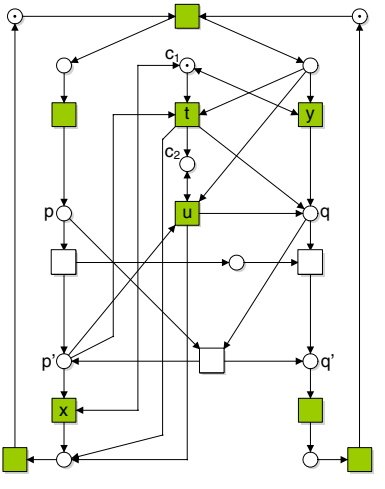
(d) MWF net M_2



(e) MWF net M_3



(f) Net $N \odot M_1$



(g) Net $N \odot M_2$

Fig. 6. Net $N \odot M_1$ is not weakly terminating while all other nets are

weak termination for MWF nets: input places contain only one token each; with more initial tokens weak termination is not guaranteed.

Another source of troubles are *causal relationships* in the refined net. Consider for example places p and q in the linear net N in Fig. 5 with initial marking $[i]$ and final marking $[f]$, which is clearly weakly terminating. The refined places p and q are causally related: the token in place p needs to be consumed before a token can be produced in place q. Refining N with the net depicted in Fig. 3(b) results in a system with a deadlock caused by the attempt to synchronize p and q. This example shows that *a reachable marking should exists in which all the places to be refined are marked to guarantee weak termination preservation.*

A first *conjecture* that it would be sufficient to check that the *synchronizing refinement of a system* (a refinement with the MWF net from Fig. 3(b)) *is weakly terminating* does not work: Consider weakly terminating system N in Fig. 6(a) with $\Omega = \{m_0, m_0 - [c_1] + [c_2]\}$, where places p and q are marked together in some reachable marking. In this net, transitions x and y are enabled as long transition t did not fire. After firing transition u, a token from place p is needed in order to mark place q.

The refinement of p and q with the net from Fig. 3(b) results in the weakly terminating net depicted in Fig. 6(b). However, the refinement of N (Fig. 6(f)) with the weakly terminating WMF net M_1 from Fig. 6(c) gives a deadlocking net: Indeed, firing sequence $t_1t_2t_3t_4t_5t_6t_7t_8t_1t_2t_9$ leads to marking $[a, b, c_2, d]$ being a deadlock. Note that firing $t_1t_2t_4t_5t_7t_8$ in N leads to marking $m_0 - [c_1] + [c_2]$, after the "control" token is moved from c_1 to c_2, a sort of causal relationship between p and q is introduced—they cannot be marked in the same marking of N any further. The synchronizing refinement does not show the deadlock: the synchronizing net prevents from moving the token from c_1 to c_2.

Another attempt to find "critical" refining MWF nets is made in Fig. 6(d) and Fig. 6(e), in which the synchronizing firing and one of the two "one-way communication" options are included, thus allowing for moving the token from c_1 to c_2. Still, the refinement of N with net M_2 (Fig. 6(g)), as well as with M_3, result in a weakly terminating net, not signalling the problem exposed in $N \odot M_1$. The root of all evil is in the "AG EF" pattern of weak termination, as it would be captured in CTL [3], where AG refers to every reachable state and EF refers to the existence of a firing sequence leading to a final marking. Due to the AG-part of the requirement, the check with the synchronizing net fails—the synchronization cuts off part of the behavior. M_2 and M_3 (partially) lift the problem, but they give too much freedom for the EF part.

These examples suggest to check that N can only produce and consume tokens to/from the refined places in a certain order: a transition can only produce a "second" token in a place from the set of the refined places (with the "first" token already consumed from it) after the other refined places have been emptied. The LTS as depicted in Fig. 7 with both solid and dashed arcs describes the desired behavior for the set of two refined places. Action p (q) indicates the production of a token in place p (q), action p' (q') indicates the consumption of a token from place p (q). For readability reasons, each state is annotated with two sets

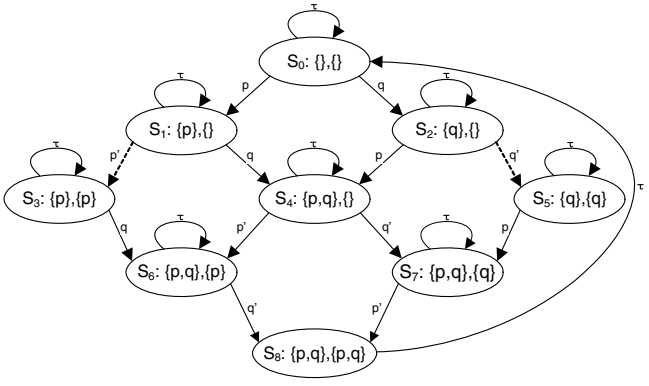


Fig. 7. May/Exit transition system for synchronizable places $\{p, q\}$

of places: the first set indicates the places that have been marked in the current iteration, the latter indicates the places from which the token already has been consumed. The initial state is s_0.

Without loss of generality, we assume that each transition in the component performs at most one action of the LTS, implying that the refined places are "disconnected": any transition is connected once to at most one place being refined. Note that using Murata reduction rules [13] in the reverse direction, as refinements, any Petri net can be transformed to a net in which a given set of places becomes disconnected, thus this requirement does not restrict the applicability of our approach.

Definition 5 (Disconnected places). *Let $N = (P, T, F)$ be a Petri net. A set of places $R \subseteq P$ is called* disconnected *if $\forall r, s \in R : r \neq s \Rightarrow ({}^\bullet r \cup r^\bullet) \cap ({}^\bullet s \cup s^\bullet) = \emptyset$ and $\forall r \in R : {}^\bullet r \cap r^\bullet = \emptyset$.*

Now let us learn from the example in Fig. 6(a) to arrive to an idea that, as we will show later, provides a sufficient condition for weak termination preservation. To see that places p and q cannot be in the set of the refined places together, we need to observe all possible behaviour, including moving the token from c_1 to c_2, and this is supported by the LTS. The reason of the deadlock we have observed is that after reaching marking $m_0 - [c_1] + [c_2]$ we are not able to reach a marking where p and q are *both* marked. This suggests to check that from every reachable non-final state of N, some final state can be reached using only the solid transitions in Fig. 7, which is not the case for N. In the next section, we present a formalism allowing for this feature and show a sufficient condition, the principle idea of which is that for the AG-part of soundness all transitions of the LTS in Fig. 6(a) may be used, whereas for the EF-part of soundness, only the solid-line transitions can be used.

5 Formalization of Synchronizable Places

We first introduce the notion of a *may/exit transition system* with two kinds of transitions: *exit transitions*, depicted with solid lines, to model the behavior needed to guarantee termination, and *may transitions*, depicted with dashed lines, to model the behavior which is allowed but not necessary to terminate. In Fig. 7, all transitions are may transitions, and the transitions depicted with solid lines are also exit transitions. As for LTSs, we assume a set of visible actions, $\mathcal{A}$ and a silent action $\tau \notin \mathcal{A}$.

Definition 6 (May/exit labeled transition system). *A* may/exit labeled transition system (MELTS) *is a 6-tuple* $(S, \mathcal{A}, \dashrightarrow, \longrightarrow, s_0, \Omega)$ *where*

- S *is a set of* states*;*
- $\mathcal{A}$ *is a* set of actions*;*
- $\dashrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times S$ *is the set of* may transitions*;*
- $\longrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times S$ *is the set of* exit transitions*, such that* $\longrightarrow \subseteq \dashrightarrow$*;*
- $s_0 \in S$ *and* $\Omega \subseteq S$ *are the initial state and a set of final state, respectively.*

Let $L = (S, \mathcal{A}, \dashrightarrow, \longrightarrow, s_0, \Omega)$ be a MELTS. For $s, s' \in S$ and $a \in \mathcal{A} \cup \{\tau\}$, we write $(L : s \overset{a}{\dashrightarrow} s')$ when $(s, a, s') \in \dashrightarrow$, and $(L : s \overset{a}{\longrightarrow} s')$ when $(s, a, s') \in \longrightarrow$. We overload the notation and write $(L : s \overset{\sigma}{\dashrightarrow} s')$ or $(L : s \overset{\sigma}{\longrightarrow} s')$ for a $\sigma \in \mathcal{A}^*$ when s' can be reached (following $\dashrightarrow$ or $\longrightarrow$ respectively) from s' by some sequence $\sigma' \in (\mathcal{A} \cup \{\tau\})^*$ such that $\sigma'_{|\mathcal{A}} = \sigma$.

A MELTS is *properly terminating* if for every state reachable with may transitions there is a sequence of exit transitions leading to a final state.

Definition 7 (Proper termination of a MELTS). *A MELTS* $L = (S, \mathcal{A}, \dashrightarrow, \longrightarrow, s_0, \Omega)$ *is* properly terminating *if for each state* $s \in S$ *and sequence* $\sigma \in \mathcal{A}^*$ *such that* $(L : s_0 \overset{\sigma}{\dashrightarrow} s)$ *there are a sequence* $v \in \mathcal{A}^*$ *and final marking* $m_f \in \Omega$ *such that* $(L : s \overset{v}{\longrightarrow} m_f)$.

Next, we define the MELTS as introduced in the previous section, named Sync(R). Let $\mathcal{N}$ be a system. Given a set R of disconnected places (the set of places we want to refine) in $\mathcal{N}$, we construct the MELTS Sync(R) as follows. Each state is represented by a pair of sets (I, O): set I indicates the places that have already received a token in the current iteration, and set O indicates the places from which the tokens have already been consumed in the current iteration. For each element r in R we identify two types of actions: (1) r, which adds the element r to I and (2) r', which adds the element r to O. As the places of R can be marked only once in each iteration, action r is only enabled in a state (I, O) if r is not present in I. Similarly, action r' is allowed only if $r \in I$ but $r \notin O$. For $r \in R$, action r is an exit transition, and action r' is an exit transition in a state (I, O) if I equals R; otherwise, it is a may transition. After each place of R has received and lost again a token, the state (R, R) is reached and the system returns with a silent step to the initial state, ready to another iteration. It is possible to stay at the same state performing silent actions. Fig. 7 shows the MELTS for a disconnected set of two places p and q.

Definition 8 (Sync(R)). *For a set R (of places), we define MELTS Sync(R) $= (S, \mathcal{A}, \dashrightarrow, \longrightarrow, (\emptyset,\emptyset), \{(\emptyset,\emptyset)\})$ by:*

- $\mathcal{A} = \bigcup_{r\in R}\{r, r'\}$;
- $S = \{(I,O) \mid O \subseteq I \subseteq R\}$;
- $\longrightarrow = \{((I,O), r, (I',O)) \mid (I,O) \in S, r \in R \setminus I, I' = I \cup \{r\}\}$
  $\cup \{((R,O), r', (R,O')) \mid (R,O) \in S, r \in R \setminus O, O' = O \cup \{r\}\}$
  $\cup \{((I,O), \tau, (I,O)) \mid (I,O) \in S, I \subset R\} \cup \{((R,R), \tau, (\emptyset,\emptyset))\}$;
- $\dashrightarrow = \longrightarrow \cup \{((I,O), r', (I,O')) \mid I \subset R, r \in I \setminus O, O' = O \cup \{r\}\}$;

Corollary 9 (Proper termination of Sync(R)). *For any set R, MELTS Sync(R) as defined in Def. 8 is properly terminating.*

Each visible transition in Sync(R) corresponds to the production or consumption of a token in one of the places of R. Since we restrict our attention to sets of disconnected places, a transition either produces a token, or consumes a token from such a place. We define a mapping function h from the transitions of N to the may/exit transitions of Sync(R) that expresses this relation.

Definition 10 (Transition mapping). *Let $N = (P, T, F)$ be a Petri net, $R \subseteq P$ be a set of disconnected places and Sync(R) as defined in Def. 8. We define the function $h_{N,R} : T \to \mathcal{A}$ for every $t \in T$ by*

$$h_{N,R}(t) = \begin{cases} r & \text{if } r \in R \cap t^\bullet, \\ r' & \text{if } r \in R \cap {}^\bullet t, \\ \tau & \text{otherwise.} \end{cases}$$

We lift the notation to sequences: for a sequence $\sigma \in T^*$ of length n, $h_{N,R}(\sigma) = \langle h_{N,R}(\sigma(1)), \ldots, h_{N,R}(\sigma(n))\rangle$. If the context is clear, we omit the subscript.

We call a set R of places in a system $\mathcal{N}$ *synchronizable*, if there is a kind of refinement relation between Sync(R) and $\mathcal{N}$, namely every firing sequence of $\mathcal{N}$ can be mapped onto a may-sequence of Sync(R), covering the AG-part of weak termination; to cover the EF-part of weak termination, for every reachable marking m of $\mathcal{N}$ there should be a firing sequence leading to a final marking corresponding to some exit sequence in Sync(R). If this requirement is met, we say that the places of R are synchronizable.

Definition 11 (Synchronizable places). *Let $\mathcal{N} = (N, m_0, \Omega)$ be a system with $N = (P, T, F)$ and a set $R \subseteq P$ of disconnected places such that $m(r) = 0$ for all $r \in R$ and $m \in \Omega \cup \{m_0\}$. Let Sync$(R)$ be as defined in Def. 8. Set R is called* synchronizable *if:*

$$\forall \gamma \in T^*,\, m \in \mathcal{R}(\mathcal{N}) : (N : m_0 \xrightarrow{\gamma} m) \implies \exists s \in S, \sigma \in T^*, m_f \in \Omega :$$
$$(Sync(R) : (\emptyset,\emptyset) \overset{h(\gamma)}{\dashrightarrow} s) \;\wedge\; (N : m \xrightarrow{\sigma} m_f) \;\wedge (Sync(R) : s \xrightarrow{h(\sigma)} (\emptyset,\emptyset)).$$

The definition of synchronizable places has some similarities with the notions of simulation [6] and refinement [11] and it encapsulates the definition of weak termination. In the net of Fig. 6(a) the set of places $\{p, q\}$ is not synchronizable, as after firing transition t the marking in which places p and q are both marked is not reachable.

Corollary 12 (Synchronizable places imply weak termination). *Let $\mathcal{N}$ be a system with $N = (P, T, F)$ and let $R \subseteq P$ be a synchronizable set of places. Then $\mathcal{N}$ is weakly terminating.*

Furthermore, the structure of the MELTS Sync(R) ensures that synchronizable places are safe: a firing sequence leading to a marking with more than one token on some place of R has no counterpart in Sync(R), and thus would contradict the definition of synchronizable places. By definition, synchronizable places are not marked in the initial marking nor in any final marking.

Corollary 13 (Synchronizable places are safe). *Let $\mathcal{N} = (N, m_0, \Omega)$ be a system with $N = (P, T, F)$. Let $R \subseteq P$ be a set of synchronizable places. Then each place $r \in R$ is safe.*

6 Synchronizable Places Preserve Weak Termination

In this section, we prove that weak termination is preserved through refinements of sets of synchronizable places. Given a system N with a set of synchronizable places R, and a weakly terminating MWF net M with a mapping function α, we need to prove that the refinement $\mathcal{N} \odot_\alpha M$ is weakly terminating.

By the definition of synchronizable places, every firing sequence of the system $\mathcal{N}$ can be replayed in Sync(R) using may transitions after projection. To relate reachable markings of N to the corresponding states of Sync(R), we introduce relation $Q_{N,R}$, defining it recursively, based on the firing rule of net N.

Definition 14 (Mapping markings to states). *Let $\mathcal{N} = (N, m_0, \Omega)$ be a system with $N = (P, T, F)$ and a set of disconnected places $R \subseteq P$, and Sync(R) be as defined in Def. 8. We define the relation $Q_{N,R} \subseteq \mathcal{R}(\mathcal{N}, m_0) \times S$ by:*

- $m_0 \, Q_{N,R} \, (\emptyset, \emptyset)$;
- *if $(N : m_0 \xrightarrow{\sigma} m \xrightarrow{t} m')$ for some $\sigma \in T^*$ and $t \in T$ and $m \, Q_{N,R} \, (I, O)$ then*
  - *if $\exists r \in (R \cap t^\bullet) \setminus I$ then $m' \, Q_{N,R} \, (I \cup \{r\}, O)$,*
  - *if $\exists r \in (I \cap {}^\bullet t) \setminus O$ and $O \cup \{r\} \subset R$ then $m' \, Q_{N,R} \, (I, O \cup \{r\})$,*
  - *if $\exists r \in (I \cap {}^\bullet t) \setminus O$ and $O \cup \{r\} = R$ then $m' \, Q_{N,R} \, (\emptyset, \emptyset)$*
  - *otherwise, $m' \, Q_{N,R} \, (I, O)$.*

If the context is clear, we omit the subscript.

To show that projecting an arbitrary firing sequence of the refinement $\mathcal{L}$ on the transitions of the original system $\mathcal{N}$ gives a firing sequence of $\mathcal{N}$, we first define a mapping φ of all reachable markings of $\mathcal{L}$ to markings of $\mathcal{N}$.

Definition 15 (Original net mapping). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4. We define the function $\varphi_{N,R} : T^* \to \mathbb{N}^{P_N}$ by*

$$\varphi_{N,R}(\sigma)(p) = \begin{cases} m(p), \text{ where } (N : m_{0_N} \xrightarrow{\sigma} m) & \text{if } p \in P_N \setminus R \\ m_0(p) + \sum_{t \in {}^\bullet p} |\sigma_{|\{t\}}| - \sum_{t \in p^\bullet} |\sigma_{|\{t\}}| & \text{if } p \in R \end{cases}$$

for all $\sigma \in T^*$. If the context is clear, we omit the subscript N, R.

The mapping ensures that given a firing sequence with which we reached a marking in the refined net, the mapped marking onto the original net is reachable as well, provided that the refining net is weakly terminating. If this is the case, then whenever the marking reached in $\mathcal{L}$ is mapped by φ to a marking N that is related to state $(\emptyset, \emptyset)$ in Sync(R), then the refining multi-workflow net is empty.

Lemma 16 (Trace inclusion for original net). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4 and M is weakly terminating. Let $(L : m_0 \xrightarrow{\sigma} m)$ for some $\sigma \in T^*$ and $m \in \mathcal{R}(\mathcal{L}, m_0)$. Then (1) $(N : m_{0_N} \xrightarrow{\sigma_{|T_N}} \varphi(\sigma))$ and (2) if $\varphi(\sigma)\, Q\, (\emptyset, \emptyset)$ then $m_{|P_M} = \emptyset$.*

Proof. We prove the first statement by induction on the structure of σ. Let $\sigma = \epsilon$. Then the statement holds by definition of $\odot$ and φ.

Now suppose $\sigma = \sigma'; \langle t \rangle$ for some $\sigma' \in T^*$, $t \in T$ and $m' \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_0 \xrightarrow{\sigma'} m' \xrightarrow{t} m)$ and $(N : m_{0_N} \xrightarrow{\sigma'_{|T_N}} \varphi(\sigma'))$. Suppose $t \in T_M$. Then $\sigma_{|T_N} = \sigma'_{|T_N}$ and $\varphi(\sigma') = \varphi(\sigma)$. Hence, the statement holds. Next, suppose $t \in T_N$. If $R \cap {}^\bullet t = \emptyset$, then the statement directly follows from the firing rule of Petri nets and the marking equation. Otherwise, i.e., an $r \in R \cap {}^\bullet t$ exists, then by the second requirement of Def. 2, $\varphi(\sigma)(r) = 1$ since otherwise M would not be weakly terminating. Then, the statement directly follows from the firing rule of Petri nets and the marking equation.

For the second statement, we have by the synchronizability of R:

$$\exists n \in I\!N : \forall r \in R : \sum_{t \in {}_N{}^\bullet r} |\sigma_{|\{t\}}| = \sum_{t \in r^\bullet_N} |\sigma_{|\{t\}}| = n$$

Suppose $n = 0$. Then the statement holds by definition of $\odot$. Now suppose $n > 0$ and the statement holds for all $n' < n$. Let $r \in R$, $\sigma', \sigma'' \in T^*$, $m' \in \mathcal{R}(\mathcal{L}, m_0)$ and $\tilde{m}' \in \mathcal{R}(\mathcal{N}, m_{0_N})$ such that $\sigma = \sigma'; \sigma''$, $\tilde{m}' = \varphi(\sigma')$, $(L : m_0 \xrightarrow{\sigma'} m' \xrightarrow{\sigma''} m)$, $m'_{|P_M} = \emptyset$, $\tilde{m}'\, Q\, (\emptyset, \emptyset)$ and $\sum_{t \in {}_N{}^\bullet r} |\sigma''_{|\{t\}}| = 1$. Since $m'_{|P_M} = \emptyset$ and $r \in (\sigma''_{|T_N})^\bullet_N$ for all $r \in R$, we have $(M : \pi_1(E_M) \xrightarrow{\sigma''_{|T_M}} \overline{m})$ and $\pi_2(E_M) \leq \overline{m}$ for some marking $\overline{m} \in \mathcal{R}(M, \pi_1(E_M))$. By Lm. 3, $\overline{m} = \pi_2(E_M)$. Since $\varphi(\sigma)\, Q\, (\emptyset, \emptyset)$, for all $r \in R$, $r \in {}_N{}^\bullet \sigma''_{|T_N}$. Hence, $m_{|P_M} = \emptyset$. □

Corollary 17. *If R is a set of synchronizable places then $Q_{N,R}$ is a functional relation.*

Note that relation Q is not a simulation relation [6]. Consider the example in Fig. 8. In this net, places p and q are synchronizable. Net N is refined with the net in Fig. 3(b). In the original net, after firing transition t_1 marking $[p, a]$ is reached and transition t is enabled. However, in the refined net, transition t cannot become enabled before transition t_2 has fired. Hence, no simulation relation exists.

We prove instead that we have the trace inclusion of the refined net into the traces of the original net when transitions of the refining net are considered as silent. A similar statement can be made for the refining MWF net M when the

(a) Net N (b) Net N refined with Fig. 3(b)

Fig. 8. Relation Q of Def. 15 is not a simulation relation

transitions of N are considered as silent. Here extra care should be taken in order to supply proper initial and final markings (e.g. the firing of t_1 in the refined net depicted in Fig. 8 results in marking $[p, a]$; its projection on M is $[p]$ not being reachable in M). Therefore, all places of R that have not yet been marked in the current iteration (i.e., the places of $R \setminus I$) are added to the marking of M, as well as the tokens that already have been removed from the final marking of M (i.e., the places of O).

Definition 18 (Refining net mapping). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4. We define the function $\psi_{N,R} : T^* \to \mathbb{N}^{P_M}$ by*

$$\psi_{N,R}(\sigma) = m_{|P_M} + \left(\sum_{r \in R \setminus I} \pi_1(\alpha(r)) \right) + \left(\sum_{r \in O} \pi_2(\alpha(r)) \right)$$

where $\varphi(\sigma)\, Q\, (I, O)$ and $m \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_0 \xrightarrow{\sigma} m)$ for all $\sigma \in T^*$. If the context is clear, we omit the subscript N, R.

Lemma 19 (ψ maps $\mathcal{R}(\mathcal{L}, m_0)$ to $\mathcal{R}(M, \pi_1(E_M))$). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4 and M is weakly terminating. Let $(L : m_0 \xrightarrow{\sigma} m)$ for some $\sigma \in T^*$ and $m \in \mathcal{R}(\mathcal{L}, m_0)$. Then $\psi(\sigma) \in \mathcal{R}(M, \pi_1(E_M))$.*

Proof. Let $\tilde{m} = \varphi(\sigma)$ and $O \subseteq I \subseteq R$ such that $\tilde{m}\, Q\, (I, O)$. We prove the lemma by induction on the structure of σ. If $\sigma = \epsilon$, the statement holds by Lm. 16.

Now suppose $\sigma = \sigma'; \langle t \rangle$ for some $t \in T$, $\sigma' \in T^*$ and marking $m' \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_0 \xrightarrow{\sigma'} m' \xrightarrow{t} m)$ and $\psi(\sigma') \in \mathcal{R}(M, \pi_1(E_M))$. Let $\tilde{m}' = \varphi(\sigma')$ and let $O' \subseteq I' \subseteq R$ such that $\tilde{m}'\, Q\, (I', O')$. Then $I' \subseteq I$ and $O' \subseteq O$. If $t \in T_M$, then ${}^\bullet t \leq m'_{|P_M}$, then $t^\bullet \subseteq P_M$ and the statement holds. Otherwise, assume $t \in T_N$. We need to do a case analysis based on the postset of transition t.

If $R \cap {}_N^\bullet t = R \cap t_N^\bullet = \emptyset$, then $I = I', O = O'$ and $m_{|P_M} = m'_{|P_M}$. Hence, the statement holds.

If $R \cap t_N^\bullet \neq \emptyset$, then a $r \in R$ exists such that $r \in t_N^\bullet$. Further, $r \notin I'$, $O = O'$ and $I = I' \cup \{r\}$, since otherwise R could not be a set of synchronizable places. By the firing rule of Petri nets, $m_{|P_M} = m'_{|P_M} + [\pi_1(\alpha(r))]$. In this way, we have:

$$\begin{aligned}\psi(\sigma') &= m'_{|P_M} + \sum_{r\in R\setminus I'} \pi_1(\alpha(r)) + \sum_{r\in O'} \pi_2(\alpha(r)) \\ &= m'_{|P_M} + [\pi_1(\alpha(r))] + \sum_{r\in R\setminus I} \pi_1(\alpha(r)) + \sum_{r\in O} \pi_2(\alpha(r)) \\ &= m_{|P_M} + \sum_{r\in R\setminus I} \pi_1(\alpha(r)) + \sum_{r\in O} \pi_2(\alpha(r)) \;=\; \psi(\sigma)\end{aligned}$$

Thus, the statement holds. A similar argument holds if $R \cap {}^{\bullet}_{N}t \neq \emptyset$. □

As a consequence of Lm. 16 and Lm. 19, boundedness is preserved by the refinement of synchronizable places.

Corollary 20 (Refinement preserves boundedness). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4 and M is weakly terminating. If $\mathcal{N}$ is k-bounded and $\mathcal{M}$ is l-bounded, then $\mathcal{L}$ is $\max(k,l)$-bounded.*

Lm. 19 implies that for every reachable marking m of $\mathcal{L}$ corresponding to a marking of $\mathcal{N}$ in which all synchronizable place are marked, there is a firing sequence from m in $\mathcal{L}$ using transitions of M only and leading to a marking in which all places of M are empty except for the final places. Note that in the refined net, one of the final places can already be emptied.

Corollary 21 (Completing trace of refining net). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4 and M is weakly terminating. Let $(L : m_0 \xrightarrow{\sigma} m)$ for some $\sigma \in T^*$, $m \in \mathcal{R}(\mathcal{L}, m_0)$, and let $s \in \{(R,O) \mid O \subseteq R\}$ be such that $\varphi(\sigma)\, Q\, s$. Then there are a marking $m' \in \mathbb{N}^P$ and a firing sequence $\nu \in T_M^*$ such that $(M : \psi(\sigma) \xrightarrow{\nu} f_M)$, $(L : m \xrightarrow{\nu} m')$ and $m'_{|P_M} \leq f_M$.*

To prove that the refinement of a set of synchronized places R preserves weak termination, we first show that for any reachable marking in the refined net that is related to the initial state of Sync(R), $(\emptyset,\emptyset)$, a firing sequence exists that reaches a final marking of the refined net.

Lemma 22 (Completing trace in refined net). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ as defined in Def. 4 such that M is weakly terminating. Let $\gamma \in T^*$ and $m \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_0 \xrightarrow{\gamma} m)$ and $\varphi(\gamma)\, Q\, (\emptyset,\emptyset)$. Then a $\sigma \in T^*$ and an $f \in \Omega$ exist such that $(L : m \xrightarrow{\sigma} f)$.*

Proof. Let $\tilde{m}_0 = \varphi(\sigma)$. Then $\tilde{m}_0 \in \mathcal{R}(\mathcal{N}, m_{0_N})$. Since R is a set of synchronizable places, there exists a firing sequence $\mu \in T^*$ with $|\mu| = n$ and marking $f_N \in \Omega_N$ such that $(N : \tilde{m}_0 \xrightarrow{\mu} f_N)$ and $(\text{Sync}(R) : (\emptyset,\emptyset) \xrightarrow{h(\mu)} (\emptyset,\emptyset))$, i.e., firing sequence μ only uses exit transitions in Sync(R). Since M is sound, a $\nu \in T_M^*$ exists such that $(M : \pi_1(E_M) \xrightarrow{\nu} \pi_2(E_M))$. Let $\tilde{m}_1, \ldots, \tilde{m}_n \in \mathcal{R}(\mathcal{N})$ such that $\tilde{m}_n = f_N$ and $(N : m_{i-1} \xrightarrow{\mu(i)} m_i)$ for all $1 \leq i \leq n$.

We construct sequence $\sigma = \sigma_1; \ldots; \sigma_n$ as follows:

$$\sigma_i = \begin{cases} \langle \mu(i) \rangle; \nu & \text{if } \mu(i)^{\bullet} \cap R \neq \emptyset \text{ and } \tilde{m}_i\, Q\, (R,\emptyset) \\ \langle \mu(i) \rangle & \text{otherwise} \end{cases}$$

Next, we need to prove that σ is a firing sequence of $\mathcal{L}$. We prove this by showing for all $1 \leq i \leq n$ the existence of markings $m_{i-1}, m_i \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_{i-1} \xrightarrow{\sigma_i} m_i)$, $\varphi(\sigma_1; \ldots; \sigma_i) = \tilde{m}_i$ and if $\tilde{m}_i \, Q \, (R, O)$ for some $O \subseteq R$ then $m_{|P_M} \leq \pi_2(E_M)$.

Suppose $n = 0$. Then $\sigma = \epsilon$. Choose $m_0 = m$. Then the statement holds trivially.

Now suppose $0 < i < n$ and a marking $m_{i-1} \in \mathcal{R}(\mathcal{L}, m_0)$ exists such that $(L : m \xrightarrow{\sigma'} m_{i-1})$ and $\varphi(\sigma') = \tilde{m}_{i-1}$ where $\sigma' = \sigma_1; \ldots \sigma\sigma_{i-1}$. Let $t = \sigma_i(1)$. Then ${}^\bullet t \leq \tilde{m}_{i-1}$, since $(N : \tilde{m}_{i-1} \xrightarrow{t} \tilde{m}_i)$. If $R \cap {}^\bullet t = \emptyset$, then $t \leq m$, and thus a $m' \in \mathbb{N}^P$ exists such that $(L : m_{i-1} \xrightarrow{t} m')$. Otherwise, an $r \in R$ exists such that $R \cap {}^\bullet t = \{r\}$. Then $\tilde{m}_{i-1} \, Q \, (R, O)$ for some $O \subseteq R$. Hence, $m_{|P_M} \leq \pi_2(E_M)$. Since ${}^\bullet t \leq \tilde{m}_{i-1}$, we have $\tilde{m}_{i-1}(r) = 1$, and hence, $m(\pi_2(\alpha(r))) = 1$. Thus, a marking $m' \in \mathbb{N}^{P_L}$ exists such that $(L : m_{i-1} \xrightarrow{t} m')$. Then $\varphi(\sigma) = \tilde{m}_i$. If $|\sigma| = 1$, choose $m_i = m'$. Then the statement holds. Otherwise, i.e., $|\sigma| > 1$, we have $\tilde{m}_i \, Q \, (R, \emptyset)$. Since $R \cap t^\bullet \neq \emptyset$, $m'_{|P_M} = \pi_1(E_M)$. Hence, a marking m_i exists such that $(L : m' \xrightarrow{\mu} m_i)$, $\varphi(\sigma) = \tilde{m}_i$ and $m_{i|P_M} = \pi_2(E_M)$. Thus, the statement holds.

Hence, σ has the desired property. □

To prove that the refinement of a set of synchronized places R preserves weak termination, we first show that from any reachable marking in the refined net, it is possible to reach a marking that corresponds to the initial state of Sync(R).

Now, we use the above lemma to show that from any marking reachable in $\mathcal{L}$ a final marking is reachable.

Theorem 23 (Refinement of synchronizable places preserves weak termination). *Let $\mathcal{L} = \mathcal{N} \odot_\alpha M$ be as defined in Def. 4 and M is weakly terminating. Then $\mathcal{L}$ is weakly terminating.*

Proof. Let $\gamma \in T^*$ and $m \in \mathcal{R}(\mathcal{L}, m_0)$ such that $(L : m_0 \xrightarrow{\gamma} m)$. We need to show the existence of a sequence $\sigma \in T^*$ and marking $f \in \Omega$ such that $(L : m \xrightarrow{\sigma} f)$.

Define $\tilde{m} = \varphi(\gamma)$. Then $\tilde{m} \, Q \, (I, O)$ for some $O \subseteq I \subseteq R$. Since R is a set of synchronizable places, a firing sequence $\sigma_1 \in T_N^*$ and marking $\tilde{m}_1$ exist such that $(N : \tilde{m} \xrightarrow{\sigma_1} \tilde{m}_1)$ and $(\text{Sync}(R) : (I, O) \xrightarrow{h(\sigma_1)} (R, O))$, i.e., σ_1 corresponds to a firing sequence of only exit transitions in Sync(R).

Then $\tilde{m}_1 \, Q \, (R, O)$ and $R \cap {}^\bullet\sigma_1 = \emptyset$, since otherwise $h(\sigma)$ was not a firing sequence in Sync(R). Then σ_1 is also a firing sequence in $\mathcal{L}$. Thus, a marking $m_1 \in \mathbb{N}^P$ exists such that $(L : m \xrightarrow{\sigma_1} m_1)$ and $\tilde{m}_1 = \varphi(\gamma; \sigma_1)$.

By Cor. 21, a firing sequence $\sigma_2 \in T_M^*$ and marking $m_2 \in \mathbb{N}^P$ exist such that $(L : m_1 \xrightarrow{\sigma_2} m_2)$, $m_{2|P_M} \leq \pi_2(E_M)$ and $\tilde{m}_1 = \varphi(\gamma; \sigma_1; \sigma_2)$.

Again since R is a set of synchronizable places, a firing sequence $\sigma_3 \in T_N^*$ and marking $\tilde{m}_2 \in \mathbb{N}^{P_N}$ exist such that $(\tilde{m}_1 : \sigma_3 \xrightarrow{\tilde{m}_2})$ and $(\text{Sync}(R) : (R, O) \xrightarrow{h(\sigma_3)} (\emptyset, \emptyset))$. Since $m_{2|P_M} \leq \pi_2(E_M)$, a marking $m_3 \in \mathbb{N}^P$ exist such that $(L : m_2 \xrightarrow{\sigma_2} m_3)$ and $\tilde{m}_2 = \varphi(\gamma; \sigma_1; \sigma_2; \sigma_3)$.

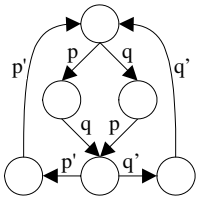


Fig. 9. LTS of places that are objective

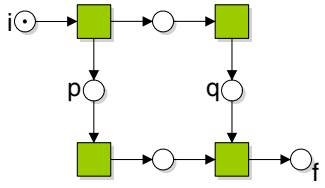


Fig. 10. Places p and q are synchronizable

By Lm. 22, a firing sequence $\sigma_4 \in T^*$ and marking $f \in \Omega$ exist such that $(L : m_3 \xrightarrow{\sigma_4} f)$. Thus, $\sigma = \sigma_1;\sigma_2;\sigma_3;\sigma_4$ has the desired property. Hence, $\mathcal{L}$ is weakly terminating. □

Clearly, synchronizability of a set of places can be effectively checked for bounded systems $\mathcal{N}$. Since weak termination can be reformulated in terms of home spaces, and the home space problem is decidable [4], we expect that the synchronizability problem is decidable for unbounded Petri nets as well.

7 Related Work

Refinements and reduction rules were in focus of the Petri nets community for a long time (see e.g. [2,13,14]). A number of rules were developed for popular subclasses of Petri nets (see e.g. [5] for reductions of free choice Petri nets, based on linear algebraic properties of these nets). The refinement we presented is a generalization of the refinement from [8] for the case of sets of places instead of a single place. In this sense our refinement is similar to the refinement $N|||_P M$ of a host net N with a daughter net M [19], but $|||_P$ is defined differently from our $\odot_\alpha$, namely, $|||_P$ basically fuses places from P in N_1 and N_2. Moreover, the focus of [19] is on the characterization of external equivalences (shown to be undecidable), while our focus is at the preservation of weak termination through refinements.

The notion of synchronizability of places is closely related to the notion of *objectivity* [15] in condition/event systems, which are Petri nets with safe places. There process semantics is used instead of interleaving semantics, and the run of a system is specified by an occurrence net, which is a, possibly infinite, acyclic marked graph. A process maps the nodes of the occurrence graph on the nodes of the condition/event system. As the net is acyclic, relation $<$ can be defined on the nodes of the net as the transitive irreflexive closure of the flow relation.

Objective places need both to be marked before they can get unmarked. The notion of objectivity can be described by the LTS shown in Fig. 9. This LTS is a subgraph of the MELTS Sync (Def. 8). If two places are objective, they are also synchronizable, as the firing sequences projected on the MELTS Sync only use states s_0, s_1, s_2, s_4, s_6 and s_7. The states s_3 and s_4, i.e. the states in which place p is already unmarked but q not yet marked (or vice versa), are never reached.

On the other hand, synchronizability does not imply objectivity: Consider the example of Fig. 8(a); places p and q are synchronizable, but not objective, as ${}^\bullet q \not< p^\bullet$ (since $\neg(t < t_2)$).

Note that the synchronic distance [17] between the sets of input transitions of two arbitrary places from a set of synchronizable places is at most one. This does not provide a sufficient condition for weak termination preservation through refinement, for this condition holds for input transitions of places p and q from net N in Fig. 6(a), for which a non-weakly termination refinement exists.

May/exit transition systems, which we introduce to capture the notion of synchronizability, resemble modal transition systems with may/must transitions, and the relation we establish between a Petri net and the MELTS Sync resembles the refinement relation of [11,12]. A modal transition system L refines another modal transition system L' if all the may transitions of L are also possible in L', and all must transitions of L' are also must transitions in L. The synchronizability definition resembles the necessary condition of must-soundness from [16]; there it is shown that weak termination is preserved through all possible (data) refinements iff from any configuration that is may-reachable from the start configuration, a subset of the final configurations is must-reachable. The main difference with our approach here lies in the fact that we do not require the Petri net system to have counterparts for *all* the exit transitions of the MELTS Sync, like it is done for must transitions, thus loosing the coupling as it is imposed by the refinement relation. Consider the example depicted in Fig. 10; $\{p, q\}$ is a set of synchronizable places, although the exit transition $(\text{Sync}(R) : (\emptyset, \emptyset) \xrightarrow{q} (\{q\}, \emptyset))$ can never be taken, implying that the net is not a refinement of Sync (when "exit" is renamed to "must").

An approach for checking weak termination of refinements of pairs of places is presented in [7], where the refinement is reduced to an application of synchronous composition. The check consists of two parts: one on the original net, and one on the refining net, based on the theory of maximal controllers. We work here with *sets* of places and use a different technique in order to guarantee the preservation of weak termination for a refinement of a set of places with an *arbitrary* weakly terminating multi-workflow net.

8 Conclusions

In this paper we have defined refinements of sets of places with multi-workflows, targeted at component-based systems. We have shown that weak termination is preserved through refinements of sets of synchronizable places. We have not proven that this condition is also a necessary condition, although we have a strong belief that it is.

We plan to implement the synchronizability check for bounded Petri nets on the basis of the standard soundness check for workflow nets (see [18]), using the synchronous product of the Petri net system and the MELTS, and enforcing the EF part with the exit-path condition.

References

1. Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)
2. Berthelot, G.: Transformations and decompositions of nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 360–376. Springer, Heidelberg (1987)
3. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
4. de Frutos Escrig, D., Johnen, C.: Decidability of home space property. Technical report, Univ. de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique Report LRI–503 (July 1989) NewsletterInfo: 35
5. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
6. van Glabbeek, R.J.: The Linear Time - Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
7. van Hee, K.M., Mooij, A.J., Sidorova, N., van der Werf, J.M.E.M.: Soundness-preserving refinements of service compositions. In: Web Services and Formal Methods 10. LNCS, Springer, Heidelberg (2011)
8. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Soundness and separability of workflow nets in the stepwise refinement approach. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, p. 335. Springer, Heidelberg (2003)
9. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Generalised soundness of workflow nets is decidable. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 197–215. Springer, Heidelberg (2004)
10. van Hee, K.M., Sidorova, N., van der Werf, J.M.E.M.: Construction of asynchronous communicating systems: Weak termination guaranteed! In: Baudry, B., Wohlstadter, E. (eds.) SC 2010. LNCS, vol. 6144, pp. 106–121. Springer, Heidelberg (2010)
11. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
12. Larsen, K.G., Thomsen, B.: A modal process logic. In: Logic in Computer Science, pp. 203–210. IEEE Computer Society Press, Los Alamitos (1988)
13. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
14. Murata, T., Suzuki, I.: A method for stepwise refinement and abstraction of Petri nets. Journal of Computer and System Sciences 27(1), 51 (1983)
15. Reisig, W.: A strong part of concurrency. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 266, pp. 238–272. Springer, Heidelberg (1987)
16. Sidorova, N., Stahl, C., Trčka, N.: Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 530–544. Springer, Heidelberg (2010)
17. Suzuki, I., Kasami, T.: Three measures for synchronic dependence in petri nets. Acta Informatica 19, 325–338 (1983)
18. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnosing workflow processes using Woflan. Computer Journal 44, 246–279 (2001)
19. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992)

Modeling and Verification of a Protocol for Operational Support Using Coloured Petri Nets

Michael Westergaard* and Fabrizio M. Maggi**

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
{m.westergaard,f.m.maggi}@tue.nl

Abstract. In this paper, we describe the modeling and analysis of a protocol for operational support during workflow enactment. Operational support provides online replies to questions such as "is my execution valid?" and "how do I end the execution in the fastest/cheapest way?", and may be invoked multiple times for each execution.

Multiple applications (operational support providers) may be able to answer such questions, so a protocol supporting this should be able to handle multiple providers, maintain data between queries about the same execution, and discard information when it is no longer needed.

We present a coloured Petri net model of a protocol satisfying our requirements. The model is used both to make our requirements clear by building a model-based prototype before implementation and to verify that the devised protocol is correct.

We present techniques to make analysis of the large state-space of the model possible, including modeling techniques and an improved state representation for coloured Petri nets allowing explicit representation of state spaces with more than 10^8 states on a normal PC.

We briefly describe our implementation in the process mining tool ProM and how we have used it to improve an existing provider.

1 Introduction

In business process management, *operational support* [10] is the capacity to provide users with recommendations about actions to be taken in order to arrive at a goal. In this paper, we aim at extending an existing infrastructure for operational support to allow stateful communication between a client and a set of operational support providers. We specify our design using a coloured Petri net model to define the protocol and to get a firm idea of our requirements. Afterward, we use state space analysis to verify that the devised protocol is correct. Finally, we use the model as blueprint for implementation.

* This research is supported by the Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

** This research has been carried out as a part of the Poseidon project at Thales under the responsibility of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 169–188, 2011.

Operational support allows a number of queries: *simple*, *compare*, *predict*, and *recommend*. A client sends a query and a partial execution trace of a workflow, and gets a response to the query. In this paper we abstract away the actual contents of the queries, so suffice to say that these four types represent increasingly complex queries, where the simplest kind allows clients to ask diagnostics about the current execution (such as whether the current execution is valid) and the more complex queries allow clients to ask for an optimal strategy for a desired goal (such as completing an execution as fast as possible).

An infrastructure for operational support is already implemented in the process mining tool ProM [9]. It is shown in Fig. 1. A Client communicates with a Workflow System and with the *operational support service* (OS Service; OSS in the following). The OSS forwards requests to a number of *operational support providers* (OS Providers; providers in the following), which may implement different algorithms, and sends back replies.

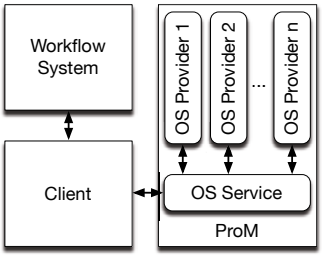


Fig. 1. Existing operational support architecture

The major problem with this implementation is that if a client provides the OSS with a trace and later with an extension, it is often possible to reuse a lot of the first computation, but as the current implementation is stateless, this is not possible. This can be addressed by making the OSS stateful, so the partial trace and results of any computation performed on it can be stored in a session. Statefulness can be implemented by each provider individually but this would be doubling a lot of effort, and we instead propose to let the OSS handle sessions for all providers, including session management such as serializing sessions on service shutdown and session garbage collection on client shutdown. Our aim is to put all complexity of session handling inside the OSS, so clients and providers remain simple (as we have more of them but only one OSS).

Our initial specification for the new operational support was as vague as "operational support with sessions". Rather than starting an implementation from this vague definition or making a more detailed textual specification, we decided to make an executable specification as a coloured Petri net (CPN) [6] model. The resulting CPN has been useful, both as a specification and as an executable prototype for verification of correctness of the devised protocol. After constructing the model, we had to address the problem that the state-space of the model was very large (exhausted memory around $10^5 - 10^6$ states). Model-alterations and implementation of a more efficient state representation allowed us to analyze the model with $10^7 - 10^8$ states, enough for this case. Modeling revealed several under-specified parts and verification revealed further problems (memory leaks and dead-locks), which were found and fixed before implementation started.

The contribution of this paper is three-fold: first, we present a viable protocol for operational support with sessions, second, we demonstrate that coloured Petri nets can be used for both making an abstract idea of a protocol more concrete and for verifying that the final design is correct even though the resulting

state-space is very large, and, third, we present a very efficient state representation for CPNs. We have spent around 1 man week on the construction of the model and around 2 man weeks on implementing a new state space analyzer.

The rest of this paper is structured as follows: in the next section, we provide the background needed to understand the rest of this paper. In Sect. 3, we present the developed model, and in Sect. 4, we sum up the analysis phase and the required fixes to the model. In Sect. 5, we outline an implementation of the devised protocol and its advantages for a real-life provider. Finally, in Sect. 6, we draw our conclusions and provide directions for future work. The reader is assumed to have a basic knowledge of coloured Petri nets but no prior knowledge of operational support is assumed.

2 Background

In this section we briefly introduce coloured Petri nets (CPNs) and the old operational support service (OSS). CPN is a high-level Petri net formalism, extending standard Petri nets. CPNs are bipartite directed graphs comprising *place*s, *transition*s and *arc*s, with inscriptions and types allowing tokens to be distinguishable. In Fig. 6 we see (part of) a CPN. Places are ovals (e.g., Working) and transitions are rectangles (e.g., Check). Places have a *type* (e.g., CLIENTxSESSIONID) and can have an *initial marking* which is a multi-set of values (tokens) of the corresponding type. Arcs contain *expressions* with zero or more free *variables*.

We call a transition and assignment of values to all its free variables a *binding element* or *binding* (e.g., Check with c=1, sid=2, t1=[] and trace=[] written Check⟨c=1,sid=2,t1=[],trace=[]⟩). An arc expression can be evaluated in a binding using the assignments, resulting in a value (e.g., (1,2) on the arc from Working to Check in the previous binding). A binding is *enabled* if all input places contain at least the tokens prescribed by evaluating the arc expressions (in the example no binding elements are enabled; if Working contained a token (1,2) and Trace a token (1, ([],[])) the binding Check⟨c=1,sid=2,t1=[],trace=[]⟩ would be). An enabled binding can *occur*, removing the corresponding tokens on input places and creating new tokens on output places (in the example, the binding would among others remove the token (1,2) from Working and move it to Waiting).

CPNs can contain multiple *module*s (or *pages*). The interface of a module is described using port places, places with an annotation In, Out, or I/O (e.g., Working is a port place and Waiting is not). A module can be represented using a *substitution transition*, which is a rectangle with a double outline (e.g., Query in Fig. 5). Places connected to a substitution transition are called *socket place*s and are connected to port places using *port/socket assignment*s. We use the convention that places in a port/socket assignment share the same name.

2.1 Operational Support Service

We have already introduced the basic operation of the old operational support service (OSS) in Sect. 1, so instead of repeating that, we focus on a single

provider, the Declare Monitor, and stress some of the problems that the old OSS encountered in this provider. In Sect. 5, we show how these issues have been addressed in the new implementation.

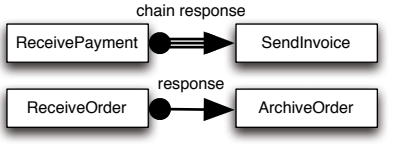


Fig. 2. Example Declare model

Monitoring Declare Models. The Declare Monitor takes as input a Declare model [8] consisting of events and constraints on the events. In Fig. 2 we see an example Declare model of a selling process. Here ReceivePayment must be directly followed by SendInvoice (expressed by the constraint chain_response) and ReceiveOrder must be eventually followed by ArchiveOrder (response).

The provider receives partial traces from clients (e.g., different executions of the selling process) and monitors the behavior of these w.r.t. the Declare model. When an event occurs, the Declare Monitor associates to each constraint of the Declare model one of the states: *satisfied*, *pending*, or *violated*. A constraint is satisfied if it is currently not violated. The pending state indicates that a constraint is currently violated but can become satisfied later. This is e.g. the case for constraints waiting for a specific event. Finally, a constraint is violated if it is currently violated and can never become satisfied again.

In Fig. 3, a client for the Declare Monitor is shown. When event ReceiveOrder initially occurs, response becomes pending (waiting for the event ArchiveOrder). ArchiveOrder never occurs in the trace so, when the execution completes, the constraint is violated. The constraint chain_response is initially satisfied. When event ReceivePayment occurs, it becomes pending (waiting for SendInvoice). When SendInvoice occurs (immediately after ReceivePayment) the constraints is satisfied again. When event ReceivePayment occurs again, it is not followed directly by SendInvoice and the constraint is violated.

The first issue identified in the existing OSS is that when a client sends a request, the OSS unconditionally sends the request to all providers, regardless of whether they can handle it or not. Second, any information needed by the provider to perform analysis must be specified when the provider is started, so clients cannot customize a provider. For instance, in the Declare Monitor, all the clients use the Declare model specified when the provider is started, so a single server cannot monitor different processes. Third, the old protocol is state-less, so a provider receives all information related to a client in each request. Therefore, the Declare monitor is forced to check the entire partial trace statically every

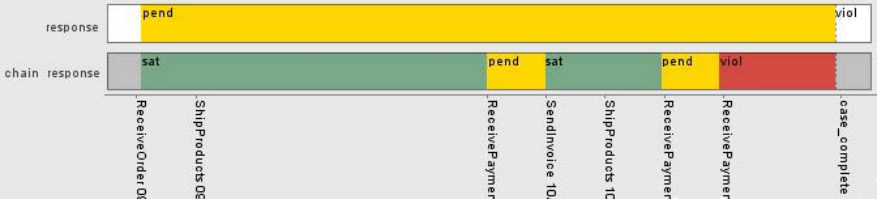


Fig. 3. Declare Monitor client

time, making run-time monitoring impossible (and impractical to simulate as the start-up cost of the Declare monitor is very large). Finally, clients cannot inform providers that an execution is completed. For the Declare Monitor, this information is crucial to report pending constraints as violated when execution ends.

3 Model of the New Protocol

We wish to extend the existing operational support service (OSS) so it supports sessions with persistent data between calls from the same client instance. We want to preserve a loose coupling between client and provider, but we still want to be able to make an intelligent pairing, so a client instance and a provider are only paired if the provider is able to provide meaningful responses to queries from the client.

The basic idea of the protocol is that a client sends a CREATE_SESSION message to the OSS. The OSS queries all available providers if they wish to be attached to the session, and the OSS responds back to the client with a SESSION_CREATED message. The client now sends CHECK messages connected to a session, which are distributed to all providers attached to the session, and an aggregate result is returned to the client in a CHECKED message. A client can send a CLOSE_SESSION message to the OSS, which is distributed to all providers registered with the session, and finally a SESSION_CLOSED message is returned to the client. Providers may register themselves with the OSS by a REGISTER_PROVIDER message and deregister themselves with a SHUTDOWN_PROVIDER message. This can happen at any time.

Our focus with this model is on the correctness of the protocol with respect to session management, so we have abstracted away any data not directly related to this and made any decision based on data using a non-deterministic choice. We have build a more detailed model than explained in this paper (explicitly modeling data and also a backwards compatibility layer). We can switch features on and off to reflect the different uses of the model (specification and verification) and for simplicity have chosen to just explain the model used for verification here. We have chosen to model all processes as terminating (so, for example, once a client instance has terminated, it will never again become alive to send requests). The model is rather large and consists of 25 modules and a total of 134 places and 65 transitions.

The top level of our model is shown in Fig. 4. We have clients to the left and the operational support server to the right (the server comprises the OSS and all providers). They communicate via two places, one for sending requests from the client to the server and one for getting replies from the server. Requests are tied to a specific client, effectively modeling

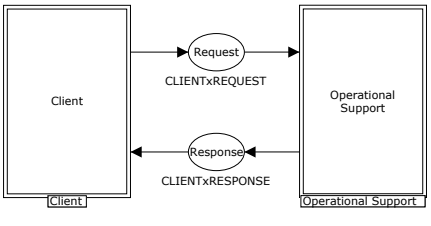


Fig. 4. Operational support model

a bidirectional channel between each client and the server. We assume that no communication channels lose packets. For most channels we allow participants to consume any transmitted packet in arbitrary order (more on this later). Our focus is on the operational support server, but we briefly introduce the client part to get a better understanding of the environment of the server.

3.1 Client

The model of the clients is shown in Fig. 5. We have folded all clients into a single net, so all places have a CLIENT component identifying the client. Clients are assumed to be single-threaded (or at least synchronize around communication with the server, so each has at most one outstanding message). Clients start in the Idle state and can Start Session by sending a CREATE_SESSION message to the server, receiving a SESSION_CREATED message back and transitioning to the Working state. In the Working state, a client can either Execute tasks internally or it can perform a Query to the operational support server. Executing tasks naturally updates our local view of the execution Trace, and sending a Query accesses the Trace as well as the communication channels to the server. When we Close Session, aside from notifying the server, we can either go to the Off-line state or the Done state. In the Off-line state we can continue Execute tasks, but we can no longer make queries. From the Off-line state we can Start Session anew and get back to the Working state. In the Done state, the client does no more work, and all messages from the server are just Discarded.

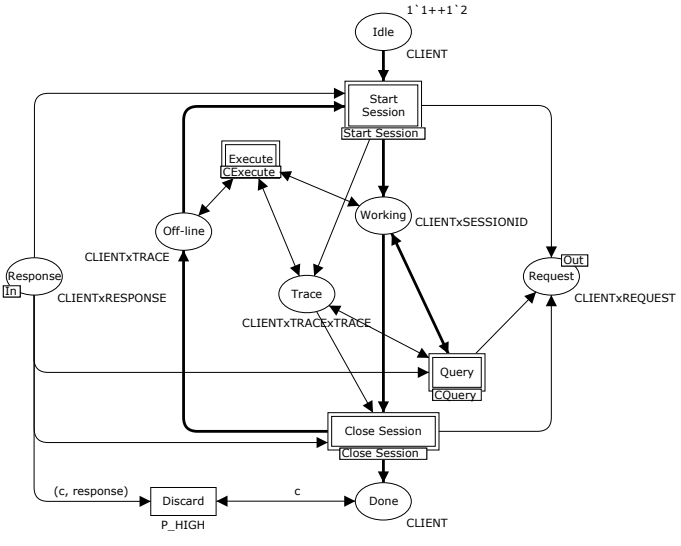


Fig. 5. Client

We have modeled all types of queries as a single action Check in the query module (Fig. 6). The Check transition reads the Trace and transmits a CHECK message to the server, containing the id of the session, the part of the trace currently not sent to the server, and a query (here just modeled as the value WHATEVER as we are not interested in the actual contents but want to model that information is sent). The Trace actually contains two traces: one for events that have not yet been seen by the server and one for events that have already been seen by the server. When we send a message to the server, we move all events from the first to the latter and only send new events. After executing a query, we go to the Waiting state and wait for response. From this state, we can Receive Response and go back to the Working state if we receive a CHECKED message. We may also receive a SESSION_CLOSED message from the server if the server has decided to garbage collect the session. Here we also go back to the Working state so we can share the session shutdown (and optional revival) procedure with the successful case.

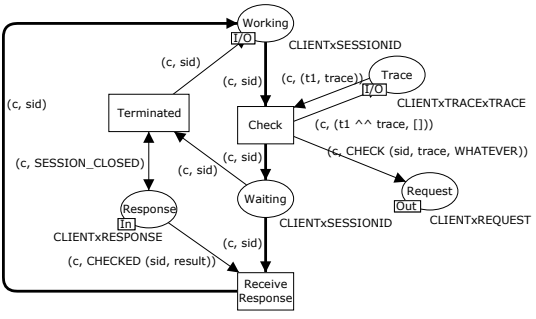


Fig. 6. Query module of client

3.2 Operational Support Server

The operational support server (Fig. 7) consists of an Operational Support Service (OSS) receiving messages from the client via Request. Requests are handled and optionally forwarded to one or more Operational Support Providers via Provider Request. Providers send back responses via Provider Response, and the OSS combines responses corresponding to a request and sends a Response back to the client containing the aggregated results from all providers. The OSS makes sure to instantiate and remove Sessions but providers can both read from and write data to sessions. A session is essentially a representation of the executed trace (the OSS makes sure this matches the values of the place Trace of Fig. 12 to the best of its knowledge) and a key/value mapping, allowing providers to store any named data.

3.3 Operational Support Service

The Operational Support Service (OSS) module is by far the most complicated. This reflects the desire to put as much functionality (and hence complexity) as possible inside this module to keep the clients and providers simple. Furthermore, we have tried to make locking as fine-grained as possible to increase concurrency. The OSS (Fig. 8) consists of two logical parts: handling communication from the client (the part of the figure to the left of places Provider Request and Provider Response) and communication from the providers (the part to the right).

The simplest part is the provider side, which handles new providers and registers them (Register Provider) as well as shuts them down (Shutdown Provider). Together these modules take care of maintaining a view of all registered providers on AllProviders. These receive input on the Provider Response channel, which is the channel used for communication from the provider to the OSS. Neither of these sends responses back to the providers as we do not expect a provider to be able to receive messages after announcing a shutdown and providers do not need any response after setup (as we assume that packets are not lost).

The client handling part performs Session Handling and handles Queries from clients. Both of these activities involve receiving messages on Request, passing them through to Provider Request, receiving responses on Provider Response and passing them on to Response. Session Handling maintains all Sessions according to requests from clients and garbage collection rules. For analysis reasons, we only allow a limited number of sessions to be created, and the number of Available Sessions keeps track of how many extra sessions we can create. We furthermore maintain a database of all currently active session ids on AllSessions (which is

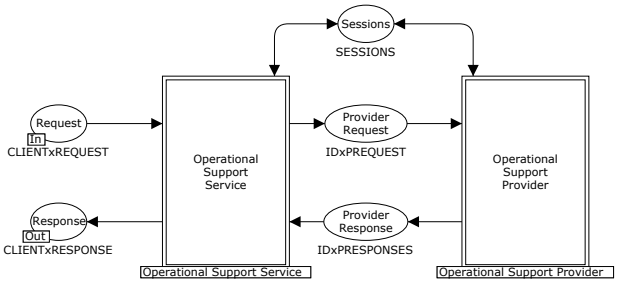


Fig. 7. Server

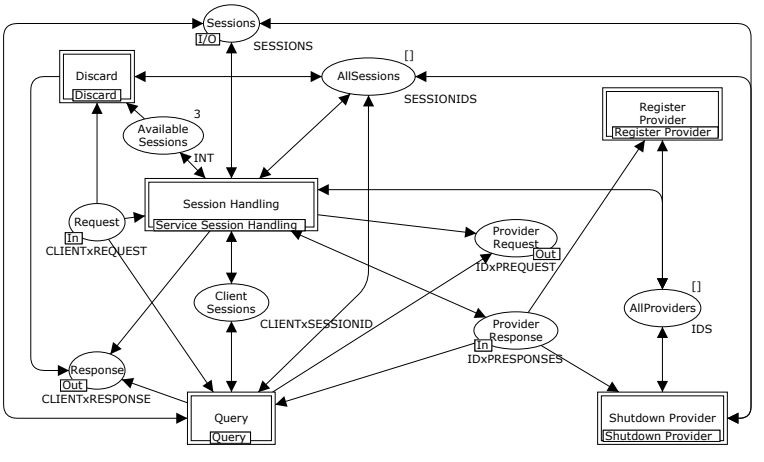


Fig. 8. Operational support service (OSS)

used to allow modules to detect when garbage collection has taken place in the middle of processing) and a mapping from client ids to session ids on Client Sessions, which is used to make sure that clients cannot hijack the session of another client. The Query module receives queries and passes them on to the correct recipient providers. The client handling part also has a Discard module, which takes care of requests from clients after we have used up all Available Sessions, and just emulates a session that is created upon request and immediately garbage collected. This makes sure that clients do not deadlock after the artificially imposed bound on the number of sessions that can be created has been reached, and would not exist in an implementation.

Session Handling. Session handling consists of three parts: session setup (Fig. 9), session tear-down (Fig. 10), and garbage collection (Fig. 11). Session setup is triggered when a CREATE_SESSION message is received. Session setup shares the layout with all main OSS modules, namely we have input from/output to the client to the left and output to/input from providers to the right, and state moves from the top to the bottom. Create Session generates a new session id (sid) using Available Sessions and adds the session to AllSessions and Client Sessions for later use. We generate a new empty session on Sessions. A session is a tuple consisting of a session id, a pair of new and old execution traces (initially both empty), and a mapping from provider id to a set of key/-value pairs, i.e., a data storage for each provider. We get all providers from AllProviders. We furthermore inform all providers that the session has been created, and move to the Pending Setup state noting the client, session id, and providers from which to expect a response. In the Pending Setup state we receive positive or

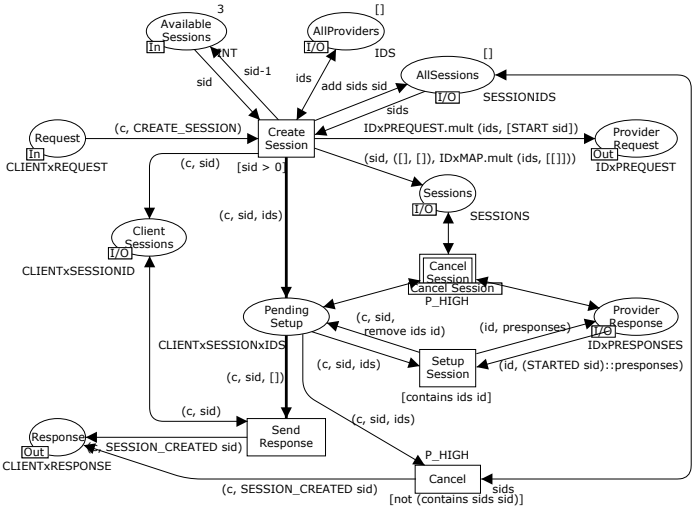


Fig. 9. Session setup module

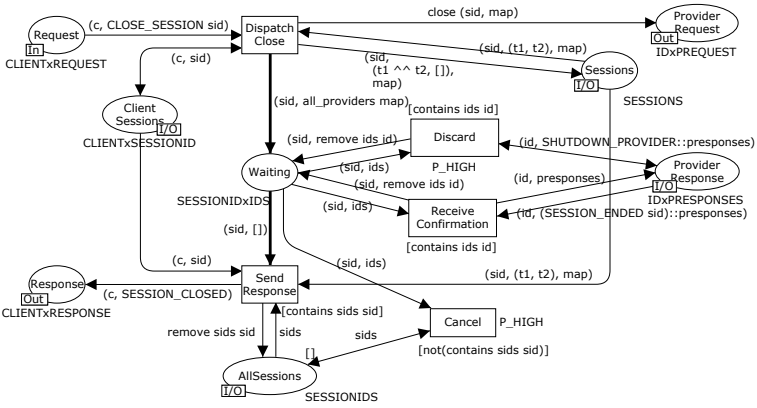


Fig. 10. Session tear-down module

negative acknowledgments from all providers associated with a session. If a provider sends a STARTED message, we successfully Setup Session for that provider, and remove it from the list of providers for which we expect a reply. If a provider rejects the session using a SESSION_ENDED message or has shut down, sending a SHUTDOWN_PROVIDER message, the session is canceled, and the entry in the session containing information about that provider is removed. The details are hidden in a module so as not to clutter the model unnecessarily. The Provider Response channel is the only channel modeled as an ordered channel to avoid session response messages building up in the channel if a provider first accepts a session and then shuts down. When the list of providers we expect a response from is empty, Send Response becomes enabled and sends a SESSION_CREATED message back to the client. The Cancel transition aborts session setup if the session has been garbage collected while waiting for response from providers. Even if we cancel a session, we send a SESSION_CREATED message to the client, so the client does not have to handle session shutdown during setup, but only during queries.

Session tear-down (Fig. 10) is handled similarly. When a CLOSE_SESSION message is received, it is matched with Client Sessions, the session trace information is updated, all providers are notified, and the OSS transitions to the Waiting state, noting all providers from which it expects a response. If a provider has been shutdown, we Discard the entry for the provider and if we Receive Confirmation we do the same. When we no longer wait for response from any providers, we remove the session from Sessions, AllSessions and Client Sessions, and send SESSION_CLOSED back to the client. As before, we can Cancel the operation if the session has been garbage collected while waiting for response.

Session garbage collection (GC) (Fig. 11) is modeled in a simple way: we GC a session if it has no more registered providers. We could make this more elaborate, e.g., allowing GC to take place if a session has a certain age.

Query. Queries are handled (Fig. 12) similarly to session handling: when the OSS receives a query message (CHECK) it checks the validity in Client Sessions, updates the trace in Sessions and dispatches to all providers. In the Waiting state it is possible to Gather Results, storing the aggregate result in Intermediate Results. Here the aggregation is just boolean or between truth values returned by providers, (in fact, for analysis, we configure the model so the produced response is always false and no aggregation actually takes place). If a provider has terminated the connection, we stop waiting for a response and Discard Request. We can also Cancel processing if the session has been garbage collected. When we have received a response from all providers, we Send Response back to the client.

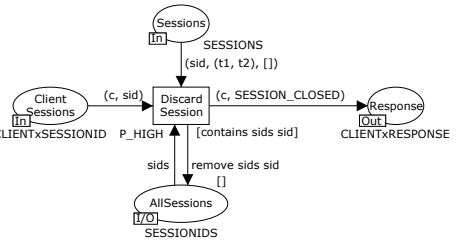


Fig. 11. Session GC module

Provider Handling. Provider handling is responsible for maintaining the AllProviders view of active providers as well as for removing provider-specific data from all sessions when a provider shuts down. When a new provider registers itself using a REGISTER_PROVIDER message (Fig. 13) it is added to AllProviders. When a provider shuts down, sending a SHUTDOWN_PROVIDER (Fig. 14), it is removed from AllProviders and all provider-specific data is scheduled to be

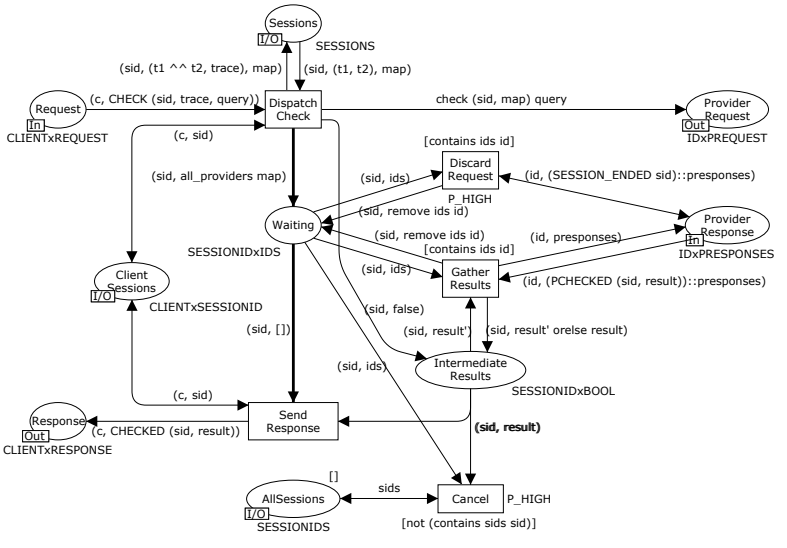


Fig. 12. Query module

removed from all sessions. As long as a session containing provider-specific data for the removed provider exists, the provider-specific data is Removed. We note that at no point do we assume that we have exclusive access to all sessions at once here. When no more sessions registered with the provider are available (a session may be shut down between the shutdown of the provider and the removal of provider-specific data from that session), the request is Done.

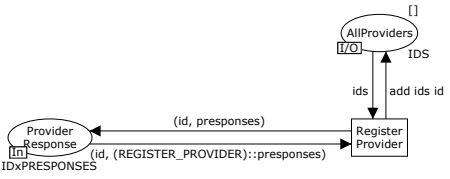


Fig. 13. Register provider module

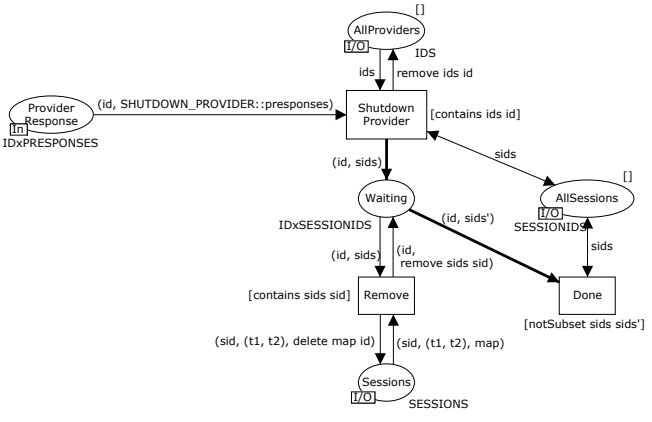


Fig. 14. Shutdown provider module

3.4 Operational Support Provider

A provider (Fig. 15) performs two administrative tasks, Provider Registration and Session Handling, and one actual task, handling Queries. Any request to a Shut Down provider is Discarded.

The provider registration module (Fig. 16) implements a simple life-cycle for providers: they are initially Stopped and after calling Register Provider and sending a REGISTER_PROVIDER message to the OSS, they are Started. Finally, they can call Remove Provider to transition to the Shut Down state and send a SHUTDOWN_PROVIDER message to the OSS.

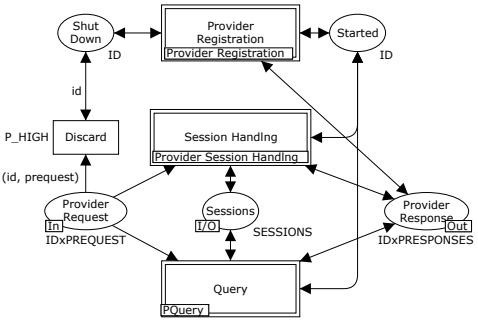


Fig. 15. Provider module

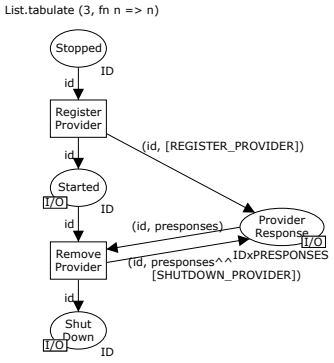


Fig. 16. Provider registration module

Session handling for providers is simple. It consists of two parts, session setup (Fig. 17) and ending sessions (Fig. 18). Setup is handled when a provider receives a START message from the OSS. If a provider is Started, it can choose to Setup Session and respond with a STARTED message. In the model, we also store the provider name in the provider-specific data of the Session. If a provider is no longer started (i.e., if it has shut down after the OSS sent the message but before it was handled), it chooses to Reject Session and reply with a SESSION_ENDED message. In fact, when a provider is Started, a non-deterministic choice is made between Setup Session and Reject Session to simulate that a provider may be incompatible with the services requested in the session. An implementation would of course not make a non-deterministic choice here, but inspect meta-data in the session request and make an informed decision. End Session (Fig. 18) is invoked when an END_SESSION message is received and sends back a SESSION_ENDED message. On termination a provider has access to the data of the session, and can use that to shut down resources referred to in the session.

Queries are handled (Fig. 19) in much the same way: When a PCHECK message is received, we access the Sessions to get provider-specific data, make sure that the provider is actually Started and send a PCHECKED message back to the OSS. The contents of the response is generated randomly as a boolean.

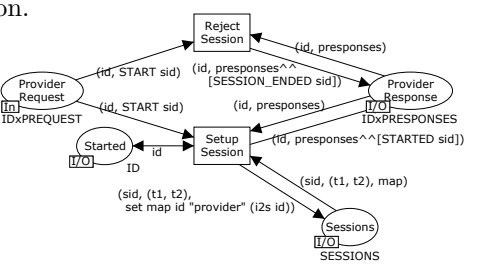


Fig. 17. Setup session module

4 Analysis

The main goal of the model developed is to serve as specification of the operational support protocol. We have tested the protocol with all features enabled using simulation but also in an iterative process using state-space analysis with

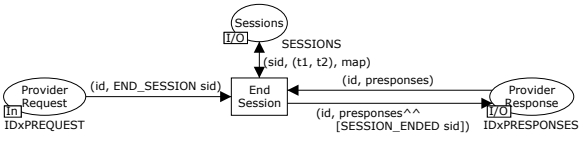


Fig. 18. End session module

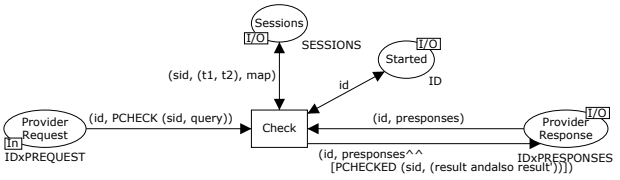


Fig. 19. Query module of provider

some features disabled. State-space analysis is a common technique for analysis of formally modeled systems as it promises fully automatic and complete coverage of all behavior of the modeled system. The method suffers from the state explosion problem, namely that the size of the state-space may be exponential or even larger in the size of the model. In this case, we also encounter the state explosion problem. We describe some techniques we have used to reduce the size of the state-space by means of model alterations, use of transition priorities, and a very efficient state-representation for general CPNs. Furthermore, we describe the errors found during analysis and how we have fixed them.

Model Alterations. Model alterations can drastically reduce the size of the state-space. In our model, the client is allowed to execute arbitrary events an arbitrary number of times (Fig. 5), causing, among others, the Trace place to be unbounded and hence the state-space to be infinite. For analysis, we therefore switch off the ability to execute events. Related to this, we have also abstracted away most data not having anything directly to do with session handling, and we have restricted the number of available sessions in Fig. 9.

Aside from standard data-abstraction, we have also used transition priorities to reduce the state-space. The idea is that sometimes tokens are produced that will have no effect on the future execution. One such example is packets that should be discarded, such as packets sent to a shut down provider in Fig. 15. Instead of non-deterministically discarding the tokens, we discard them immediately. Otherwise, we basically double the size of the state-space for each such possible token, as we have one copy of each following marking where the token is discarded immediately and one where it is discarded last (and the two copies are interconnected by discarding the token at any intermediate point). By discarding the token immediately we only have one copy, reducing the state-space. We do the same reduction for the Cancel transitions in Figs. 9, 10 and 12, and for Discard Session in Fig. 11.

New State-space Tool. We used exploratory modeling to get a hold of our requirements for the protocol. This involved using state-space exploration as a debugging technique. In order to accommodate that, we used two known techniques: translating liveness-properties to safety properties (really dead-locks) and using depth-first traversal to find errors as fast as possible and report

them on-the-fly. To support that, we could not easily use the built-in state-space generator in CPN Tools [3], as it generates the state-space in a breath-first manner and is designed around off-line verification. We could also not directly use ASAP [11] as it has no support for priorities of transitions.

Safety properties, including dead-lock freeness and bound violations, are a particularly simple kind of properties that can be determined by looking at each state in isolation. Not all interesting properties are safety properties, though. This includes a property like "any session created is eventually torn down if the client terminates", which can only be determined by looking at all (possible infinitely many) execution traces. This can be done on-the-fly but requires representing the synchronized product of the state-space and a property automaton, making it infeasible in our case. Instead we translate such interesting properties to safety properties by making sure that no entities (clients, sessions, and providers) in the model can be revived after terminating. This ensures the system has goal states (all clients and providers have shut down) and we can check if the system has dead states where not all sessions have been torn down.

As we could not use any off-the-shelf state-space generator, we made our own. We did not have complicated requirements as we only had to find dead states and provide error-traces. For this reason we decided to use Access/CPN [12], which makes it possible to access the CPN simulator used in CPN Tools from Java. The reason for not implementing this in ASAP instead is that Java compared to Standard ML (the language used in ASAP) allows finer grained control over memory, access to more memory, and support for threads.

During initial analysis, we noticed that many places share the same marking and are often empty. This prompted us to implement an efficient state representation exploiting this. The final representation is shown in Fig. 20. CPN Tools already uses a state-representation sharing on the levels of values, markings, and places [2], but it is structured, requiring several pointers for each place. Instead we propose using an unstructured representation: store all multi-sets as strings and enumerate them. We can then store the marking of a place using an integer. Instead of storing places in a structured way, we enumerate them, allowing us to represent a state using $\#places \cdot |integer|$ bits (plus the overhead of storing the multi-sets). Rather than of wasting an integer for places with empty markings, we store a bit-map indicating non-empty places, and only store a marking for them. Finally, we do not need to use an entire machine word to store an integer. Instead, we can use just 8, 16, 24, or 32 bits depending on the value of the largest integer used. We do not need more bits as, if we have more than 2^{32} different multi-sets, chances are we are not able to analyze the model anyway. We add to each state two bits indicating the number of bytes used for each integer, resulting the representation shown in Fig. 20. We translate to this representation using a map from multi-sets to their enumerated value and a counter for the next multi-set id, inspecting places in a canonical order. We translate from this representation by additionally having an array storing each multi-set in the entry with its id. This representation is generic and has also proved efficient for other models.

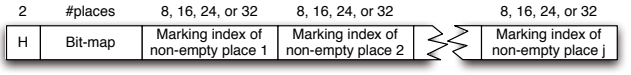


Fig. 20. Compact state representation

We currently store each state in a separate array, imposing an overhead of 16 bytes plus enough to make it a multiple of 8 bytes. In our case, we used a total of 3.18 giga-bytes to represent 68,923,926 states (including overhead and 8.8 mega-bytes for mappings between multi-sets and ids), for an average of 49.5 bytes per state. To store these, we use an additional 2–4 pointers (8–16 bytes), for at total of up to 65.5 bytes per state, allowing us to store this in just over 4 giga-bytes of memory. We can represent more than 10^8 states explicitly in memory on a standard computer with 8 giga-bytes of memory (we can use 32-bits to address up to 32 giga-bytes of memory by using packed pointers addressing objects aligned at positions divisible by 8), which we have made use of on several intermediate models. Without reduction, CPN Tools was able to generate 200,000–300,000 states before exhausting available memory. The amount of memory used by our compact representation is comparable to other memory-efficient but potentially time-consuming explicit representations of the state-space for coloured Petri nets. For example, the ComBack [4] method uses 140–150 bytes per state for realistic examples in the Standard ML implementation. In theory the ComBack method uses just 20 bytes per state but does so by imposing a quite heavy penalty on execution time. Using the sweep-line method [7] it is possible to represent a state in around 8 bytes plus 8 bytes per transition at the cost of not easily being able to reconstruct a compactly represented state, a potential cost of re-exploring parts of the state-space, and a peak memory usage which may be much larger. These methods can also gain from our compact state representation for intermediate representations and caches. Other methods for compact state representation, such as bit-state hashing [5] and hash compaction [13], do not guarantee full coverage, and there is no good approach for symbolic representation of fully general CPNs as it is not known how to represent the transition relation symbolically.

4.1 Errors Found

The non-trivial parts of the protocol consist of the interaction between client communication and session garbage collection, and handling of provider termination. For example, a scenario where a client sets up a session, initiates a query, and during the query one of the providers handling the request terminates, should be handled by the OSS. As all entities in our model are terminating (i.e., cannot be revived after stopping), this corresponds to checking that in all dead states, all entities are in their shut down state (meaning they did not dead-lock) and that in all dead states there are no sessions alive (so all sessions can be terminated correctly), no providers are registered (so the OSS does not dead-lock in an undesired state), and there are no outstanding messages on either of the

communication channels (so neither of the components dead-locked or did not process all messages correctly).

We had several cycles of modeling and analysis, and after fixing some minor initial modeling problems, we were left with three major bugs. The first bug was in the interplay between OSS communication with providers and provider shutdown. The second bug was mainly due to the fact that we modeled all communication channels without an explicit ordering, and the third bug was more serious and caused by the interplay between session garbage collection and provider communication.

The first version of our model did not consider a provider terminating in the middle of a query or session tear-down, leading to dead-locks in the OSS (dead states where the OSS was in Waiting in Figs. 10 and 12). Instead the OSS would wait indefinitely for a SESSION_ENDED which would never arrive. We therefore added Discard transitions in those cases, solving this.

The second problem arose from the fact that the protocol between the provider and the OSS is one-way. If a provider replies to a query and immediately shuts down, the OSS would have both an answer and a SESSION_ENDED message for the same provider in Provider Response in Figs. 9, 10 and 12, enabling both the Discard/Cancel Session transition and the transition for successfully receiving a response (Setup Session/Receive Confirmation/Gather Results). We could fix this by adding an explicit response for each message to the provider, but this would unnecessarily clutter the model. As we expected to implement the protocol over TCP (which ensures in-order delivery of messages), we instead modeled this channel as first-in/first-out ordered, thereby fixing the problem. To keep the model simpler, we do not impose this on the remaining channels.

Initially, we did not have Cancel transitions on the OSS components in Figs. 9, 10 and 12. If a client made a request, registered one provider, made a query, and then the provider shut down (causing the session to be garbage collected), the OSS would be stuck in the Waiting state. This third problem can occur in various forms for all three modules, and is fixed by adding the AllSessions place maintaining a view of all sessions not garbage collected and using that information to Cancel a request if necessary. Our initial solution had the Cancel transitions explicitly notify clients that a request was canceled, but this could lead to stale messages in the communication channels if garbage collection (Discard Session in Fig. 11) also sent a SESSION_CLOSED message, so that was removed.

The model presented in the previous section is the final model which has been analyzed and verified to be without errors. The presented model has 2 dead states. In them, all clients are in the Done state, all providers are Shut Down and both AllProviders and AllSessions contain empty lists. Available Sessions contains either 0 or 1, corresponding to the number of unused session identifiers in the model (each client has to consume at least one session). All other places are empty. These dead states therefore exactly exhibit the desired behavior: no clients or providers are stuck, all sessions are removed, and all channels are empty. Analysis has fixed three problems in the model; two of these would most likely also have shown themselves in an implementation and one made an implicit

assumption explicit in the model. This is of course only a single instance of the model, but we have also verified it for other configurations, and as the model has no limits pertaining to the configuration, we are confident the protocol works in all configurations.

5 New Implementation

We have implemented the model described in Sect. 3 and analyzed in Sect. 4 as a service in the process mining tool ProM [9]. The entire implementation was done by one person in two days.

A provider has to implement the interface in Figure 21. The interface shown is slightly simplified as we provide a little more meta-information for session setup and queries in the implementation, but this is not important for our discussion. The accept method is invoked whenever a client creates a session. The session corresponds to a session in the model, and the queryLanguages parameter describes which query languages will be used (not modeled). The accept method returns a boolean indicating whether the provider is willing to handle the session (corresponds to the STARTED/SESSION_ENDED messages). destroy is called when a session is shut down. The simple, comparison, predict, and recommend methods implement the four kinds of queries. They are parametrized with a result type, R, and a query language type L (a query is not necessarily a string, but can also be structured), and each method takes a session and a query as parameters as well as a boolean indicating whether the execution is done (can have impact on the result of queries; for example a Declare process may have temporarily violated constraints, which is allowed unless the execution has terminated). Providers may be called with different session parameters from different threads, so providers should not store data locally between calls, but instead store all information in the Session object.

The OSS has methods for adding and removing providers, addProvider and removeProvider, which each take a Provider as parameter. A client class exists with an interface similar to the Provider in Fig. 21.

Monitoring Declare Models. In the new OSS a client instance and a provider are paired only if, inspecting the meta-data available in the session request, the provider chooses to setup the session. This ensures a client is only bound to providers able to provide meaningful results. Meta-data sent on setting up a

```
1  public interface Provider extends Serializable {
2     boolean accept(Session session, List<String> queryLanguages);
3     void destroy(Session session);
4
5     <R, L> R simple(Session session, L query, boolean done);
6     <R, L> Prediction<R> comparison(Session session, L query, boolean done);
7     <R, L> Prediction<R> predict(Session session, L query, boolean done);
8     <R, L> Recommendation<R> recommend(Session session, L query, boolean done);
9  }
```

Fig. 21. Provider Interface

session allows clients to customize providers by setting configuration parameters. For instance, in the Declare Monitor, each client can set the Declare model which should be used for monitoring, allowing clients executing workflows from different models to use the same server.

Using sessions, a provider can store case-specific information, such as the current state of each Declare constraint. Starting from the current recorded state, the Declare Monitor is able to compute the new state for each constraint when events occur without replaying the entire partial trace each time. This is crucial for the implementation of a run-time monitor which is no longer reduced to an expensive statical checker.

In the new implementation of the OSS, a client can inform the provider that a trace is completed. The Declare Monitor can use this information to raise a violation for all those constraints which are in a pending state when the data stream completes.

6 Conclusion and Future Work

We have presented a new protocol for operational support within business process management supporting sessions, thereby alleviating problems found during development of a real-life provider. The protocol is the result of iterative prototyping and state-space analysis using coloured Petri nets. In addition to the developed protocol, we believe the techniques developed during the prototyping are generally applicable. This includes a very compact state representation for general CPN models, allowing explicit analysis of state-spaces with more than 10^8 states, and demonstrating that model alterations and use of priorities to reduce concurrency reduce the state-space sufficiently. We believe that by building a prototype using a formal model instead of a textual specification or a direct implementation has led to a much clearer and better protocol as well as much faster development. Analysis revealed two major and one minor problems in the protocol which were fixed before implementation. Using the formal model as blueprint for the implementation has made the implementation next to trivial, as evidenced by the implementation time frame of two person days.

Future work includes extending the protocol with a cross-session cache, which can, e.g., be used by the Declare Monitor to store the representation of a Declare Model used for monitoring (which is expensive to create). We do not expect this to have a major impact on the protocol. Furthermore, we have only considered a single-client/single-case scenario here, where a single client is working on a single case. It would also be interesting to consider the cases where two or more clients work on a single case, where a single client works on multiple cases, and where multiple clients work on multiple (not necessarily the same) cases. This should be possible by allowing sharing of sessions, which again would require authentication, and by extending the protocol with a means for a provider to easily consider multiple sessions at once.

It would be interesting to implement the current compact state representation in ASAP [11] to evaluate the performance of the representation without the

overhead of communication between two processes. Also, while we can reduce the overhead per state a bit more, we do not believe that we can explicitly represent states much smaller than now. It would be interesting to instead use a symbolic representation (even if states are calculated explicitly), e.g., using BDDs [1].

Acknowledgment. The authors wish to thank Marco Montali for his input to the design of the new operational support protocol.

References

1. Bryant, R.E.: Graph Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
2. Christensen, S., Kristensen, L.M.: State Space Analysis of Hierarchical Coloured Petri Nets. Petri Net Approaches for Modelling and Validation, 1–16 (2003)
3. CPN Tools webpage, `http://cpntools.org`
4. Evangelista, S., Westergaard, M., Kristensen, L.M.: The ComBack Method Revisited: Caching Strategies and Extension with Delayed Duplicate Detection. ToPNoC 3, 189–215 (2009)
5. Holzmann, G.J.: An Analysis of Bitstate Hashing. FMSD 13, 289–307 (1998)
6. Jensen, K., Kristensen, L.M.: Coloured Petri Nets – Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009)
7. Mailund, T., Westergaard, M.: Obtaining Memory-Efficient Reachability Graph Representations Using the Sweep-Line Method. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 177–191. Springer, Heidelberg (2004)
8. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: Proc. of EDOC 2007, p. 287 (2007)
9. Process mining webpage, `processmining.org`.
10. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.J.: Workflow Simulation for Operational Decision Support. Data Knowl. Eng. 68, 834–850 (2009)
11. Westergaard, M., Evangelista, S., Kristensen, L.M.: ASAP: An Extensible Platform for State Space Analysis. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, Springer, Heidelberg (2009)
12. Westergaard, M., Kristensen, L.M.: The Access/CPN Framework: A Tool for Interacting With the CPN Tools Simulator. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, Springer, Heidelberg (2009)
13. Wolper, P., Leroy, D.: Reliable Hashing without Collision Detection. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 59–70. Springer, Heidelberg (1993)

Finding a Witness Path for Non-liveness in Free-Choice Nets

Harro Wimmel and Karsten Wolf

Universität Rostock, Institut für Informatik

Abstract. One of the disablers of structural Petri net verification techniques is the lack of diagnosis information that is easily understandable. In this article, we improve this situation for a particular technique: the siphon and trap based verification of liveness in free-choice nets. Instead of the information "there is a siphon without included marked trap", we exhibit an execution path that leads from the initial marking to a marking m^* and a set of transitions that mutually block each other and are thus dead at m^*. The latter information can be much more easily comprehended by non-experts in Petri net theory. We provide experimental results suggesting that our method is competitive to related state space techniques.

Keywords: Petri Net, Free-choice, Siphon, Trap, Commoner-Hack.

1 Introduction

Liveness is one of the fundamental correctness criteria for Petri nets. If a Petri net system is live, it is well-behaved in a certain sense: Every transition can be enabled from every reachable marking, so no part of the net is unused or can be finally switched off. Liveness is investigated in various application domains, e.g. for workflows or for manufacturing systems [16,4].

Unluckily, while liveness is decidable [7,17], it is still extremely difficult to implement the corresponding routines. There is a reduction of the reachability problem for Petri nets to the question whether a net is live, and the former is known to be EXPSPACE-hard [11]. If liveness needs to be decided for a Petri net, one usually uses one of the numerous available model checkers which explore the state space of the net using either explicit or symbolic techniques to alleviate the state explosion problem. If the net is unbounded, i.e. the state space is infinite, the liveness test usually fails due to insufficient memory or runs eternally. Of course, that may even happen for bounded systems if they are large enough. Abstraction techniques may increase the number of feasible problem instances but are typically using domain specific assumptions.

If a state space based model checker finds the net to be non-live, it can provide a witness for this. Typically, such a witness consists of a path to a marking where some transitions are dead (i.e. mutally blocking each other). Exhibiting those transitions, or showing *why* they are dead is not necessarily part of the diagnosis but can be added with reasonable efforts.

There are large subclasses of Petri nets for which the liveness problem is still hard but much easier than in general. One example is the class of free-choice nets where the presets of any two transitions must be identical or disjoint. In this class, liveness

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 189–207, 2011.

is co-NP-complete, which is certainly an improvement. A theorem of Commoner and Hack [6,3] states that a free-choice net is live if and only if every of its siphons contains a marked trap, where siphons and traps are sets of places that cannot become marked/unmarked, respectively, once they are unmarked/marked. If a free-choice net is not live, we may find a siphon in it that does not contain a marked trap.

This structural situation is not an optimal witness against liveness though, in particular if it is to be presented to a non-expert in Petri net theory. It may be possible to fire all the transitions in the postset of the siphon at some time, and even mark a formerly unmarked trap inside the siphon. Of course, at least one siphon without a marked trap must remain, but since there may be exponentially many siphons in the number of places of the net, it is difficult to tell which siphon that will be.

In this paper, we shall devise an algorithm that will try to consider as few siphons as possible to find a better witness for non-liveness of a free-choice net. This witness is basically an execution sequence that transforms the initial marking into some marking m^* where one siphon is completely dead, i.e. no post transition can ever fire again. This situation requires much less theoretical understanding by a non-expert to realize non-liveness. Moreover, our approach is a step towards unifying the shape of diagnostic information between state space and structural techniques which may simplify their joint use.

In our method, finite run time even for unbounded free-choice nets is guaranteed, which is a principal advantage of our method compared to state space techniques.

In section 2, we give some basic definitions and draw some more complex conclusions on siphons and traps in section 3. Section 4 deals with unmarked but markable places and in section 5 a graph is constructed that reflects dependencies between siphons. This allows us to either empty a siphon or come to a state where every siphon needs a token from some other siphon to be able to fire any transition in its postset. In both cases, a witness is found. Section 6 shows how to extract this witness if it isn't obvious. In section 7 we take a look at the correctness of the algorithm and give experimental results for an implementation. In particular, we compare our results to our explicit state space tool [20] which has proved competitive in a number of case studies [5] and applications [15].

2 Basic Definitions

Definition 1 (Petri net, marking, firing sequence). *A* Petri net N *is a tuple* (S, T, F) *with finite sets* S *of* places *and* T *of* transitions*, where* $S \neq \emptyset \neq T$ *and* $S \cap T = \emptyset$*, and a mapping* $F\colon (S \times T) \cup (T \times S) \to \mathbb{N}$ *defining* arcs *between places and transitions. The sets* $^\bullet x = \{y \in S \cup T \mid F(y, x) > 0\}$ *and* $x^\bullet = \{y \in S \cup T \mid F(x, y) > 0\}$ *are called the* preset *and* postset *of* $x \in S \cup T$*, respectively. We allow the canonical extensions* $^\bullet X = \bigcup_{x \in X} {}^\bullet x$ *and* $X^\bullet = \bigcup_{x \in X} x^\bullet$ *for sets* $X \subseteq S \cup T$.

A marking *or* state *of a Petri net is a map* $m\colon S \to \mathbb{N}$*. A place* s *is said to contain* k tokens *under* m *if* $m(s) = k$*. The number of tokens under* m *on some set of places* $X \subseteq S$ *is denoted by* $m(X) = \sum_{s \in X} m(s)$*.* X *is* marked *under* m *if* $m(X) > 0$*, otherwise it is unmarked. A transition* $t \in T$ *is* enabled under m*,* $m[t\rangle$*, if* $m(s) \geq F(s, t)$ *for every* $s \in S$*. A transition* t fires under m and leads to m'*,* $m[t\rangle m'$*, if additionally* $m'(s) = m(s) - F(s, t) + F(t, s)$ *for every* $s \in S$.

A word $\sigma \in T^*$ is a firing sequence under m and leads to m', $m[\sigma\rangle m'$, *if either $m = m'$ and $\sigma = \varepsilon$, the empty word, or $\sigma = wt$, $w \in T^*$, $t \in T$ and $\exists m'': m[w\rangle m'' \wedge m''[t\rangle m'$. A firing sequence σ is enabled under m, $m[\sigma\rangle$, if there is some marking m' with $m[\sigma\rangle m'$. A marking m' is called* reachable *from a marking m if there is a firing sequence σ with $m[\sigma\rangle m'$. The set of all markings reachable from m is denoted by $\mathcal{R}(m, N)$ (or shorter $\mathcal{R}(m)$ if N is fixed).*

As usual, places are drawn as circles (with tokens as black dots inside them), transitions as rectangles, and arcs as arrows with $F(x, y) > 0$ yielding an arrow pointing from x to y. In case $F(x, y) = F(y, x) > 0$, we may sometimes draw a line with arrowheads at both ends. Arc weights $F(x, y) > 1$, usually denoted by the weight written next to the arc arrow, will not occur in this paper.

The structure theory of Petri nets knows about a few constructs that can help analyzing Petri nets, two of them are siphons and traps:

Definition 2 (Siphons and Traps). *Let $N = (S, T, F)$ be a Petri net. Any set $D \subseteq S$, $D \neq \emptyset$ with ${}^\bullet D \subseteq D^\bullet$ is called a* siphon *of N. Any set $Q \subseteq S$ with $Q^\bullet \subseteq {}^\bullet Q$ is called a* trap *of N.*

The exclusion of the empty set from the set of siphons is done for practical reasons only when we later talk about siphons containing marked traps. If a trap Q is marked under some marking m, the firing of a transition removing a token from Q will always also produce a token on Q. An unmarked siphon D can never obtain any tokens as any transition in ${}^\bullet D$ is also in $D^\bullet$ and thus has an empty place in its preset.

Definition 3 (Dead Places). *Let $N = (S, T, F)$ be a Petri net. A set of places $X \subseteq S$ is called* dead *under a marking m if $\forall t \in {}^\bullet X \cup X^\bullet \; \exists s \in X \cap {}^\bullet t: m(s) < F(s, t)$.*

The marking on a dead set of places remains unchanged no matter which (enabled) transitions fire. Note that for siphons, ${}^\bullet X$ can be omitted due to ${}^\bullet X \subseteq X^\bullet$. An unmarked siphon X (under m) is dead by definition.

Corollary 1 (Dead Siphons and Marked Traps). *Let $N = (S, T, F)$ be a Petri net and m a marking of N. If a trap Q is marked under m, Q is marked under every marking in $\mathcal{R}(m)$. If a siphon D is unmarked or dead under m, D is unmarked or dead under every marking in $\mathcal{R}(m)$, respectively.*

At this point we restrict ourselves to a famous subclass of Petri nets, the (extended) free-choice nets. We disallow arc weights greater than one and asymmetric conflicts between transitions from our nets, i.e. transitions competing for the same token always have the same preset.

Definition 4 (Free-choice nets [6,3]). *A Petri net $N = (S, T, F)$ is called a* free-choice net *if $F(x, y) \leq 1$ for all $x, y \in S \cup T$ and ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ implies ${}^\bullet t = {}^\bullet t'$ for all $t, t' \in T$.*

Our goal is to find witness paths, i.e. firing sequences leading to a dead or unmarked siphon, proving the non-liveness of a free-choice net, where liveness is defined as follows.

Definition 5 (Live and Dead). *Let $N = (S, T, F)$ be a Petri net and m_0 a marking. A transition $t \in T$ is called* live *at m_0 if for all $m \in \mathcal{R}(m_0)$ there is a marking $m' \in \mathcal{R}(m)$ such that $m'[t\rangle$. If t is not enabled under any reachable marking $m' \in \mathcal{R}(m_0)$, we call it* dead *at m_0. The Petri net N is called* live *or* dead *at m_0 if all its transitions are live or dead at m_0, respectively.*

A transition t is neither live nor dead at m_0 if there are markings $m, m' \in \mathcal{R}(m_0)$ such that t is enabled under m but dead at m'. Liveness means that whichever transitions have been fired so far, every transition can be enabled again. The liveness problem [7], i.e. finding out if a Petri net is live at a given marking, is at least as hard as the well-known reachability problem [12,9,10], for which no upper complexity bound is known so far. For free-choice nets however, the problem is "only" co-NP-complete [8,3] and can be formulated easily in terms of siphons and traps.

Definition 6 (Siphon-Trap-Property). *The Siphon-Trap-Property for a Petri net $N = (S, T, F)$ and a marking m is defined as*

$$\mathsf{STP}(N, m) \equiv \forall \textit{ siphon } D \subseteq S \;\; \exists \textit{ trap } Q \subseteq D\colon \; m(Q) > 0.$$

A fundamental theorem found by Commoner and Hack provides us with the connection between liveness and siphons and traps for free-choice nets.

Theorem 1 (Commoner-Hack [6,3]). *A free-choice net N is live under a marking m if and only if $\mathsf{STP}(N, m)$ holds.*

A fast decision algorithm for liveness of free-choice nets has been implemented using a reduction to the satisfiability problem of Boolean formulae and forwarding the liveness problem to a state-of-the-art SAT-solver [13]. In the case a free-choice net is not live the algorithm provides a siphon that does not contain a marked trap. While this is formally enough proof for the non-liveness, it is not intuitively clear why such a siphon leads to non-liveness. Instead, it would be better to present an unmarked (or at least dead) siphon.

3 Properties of Siphons and Traps

In this and the following sections let $N = (S, T, F)$ be a fixed free-choice net and m_0 a fixed marking of N. This section is dedicated to some more complex properties of siphons and traps. Some of the following results are folklore and mentioned e.g. in [3].

Lemma 1 (Closure under union). *The following sets are closed under union:*

- $\{D \mid D \textit{ is a siphon}\}$,
- $\{Q \mid Q \textit{ is a trap}\}$,
- $\{D \mid D \textit{ is an unmarked siphon}\}$,
- $\{D \mid D \textit{ is a dead siphon}\}$,
- $\{Q \mid Q \textit{ is a marked trap}\}$.
- $\{D \mid D \textit{ is a siphon}, \exists \textit{ marked trap } Q \subseteq D\}$,

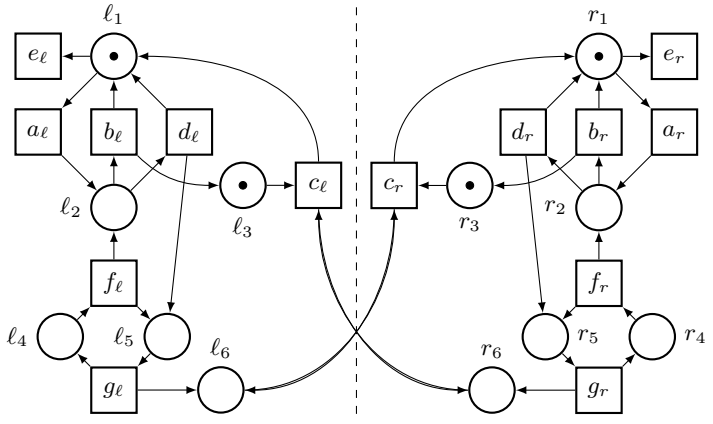


Fig. 1. A counterexample against the closure under union of siphons not containing marked traps. The places on each side of the dashed line form a siphon without marked trap, but $\{\ell_3, r_6\}$ and $\{r_3, \ell_6\}$ (and some other sets) are marked traps of the union. Note the arcs in both directions between c_ℓ/r_6 and c_r/ℓ_6

The set of siphons not containing a marked trap is not *closed under union.*

Proof. Let D_1 and D_2 be two siphons. Then, ${}^\bullet(D_1 \cup D_2) = {}^\bullet D_1 \cup {}^\bullet D_2 \subseteq D_1{}^\bullet \cup D_2{}^\bullet = (D_1 \cup D_2)^\bullet$. An analogous result holds for traps. For unmarked siphons and marked traps just note that two unmarked siphons together are still unmarked and two marked traps together are still marked. If two siphons are dead, no transition in one of their postsets is enabled, so no transition in the union of the postsets is enabled. For the last case, note that a marked trap in one of the siphons is also contained in their union.

For siphons not containing a marked trap a counterexample (in form of a free-choice net) is presented in Fig. 1. The places on each side of the dashed vertical line form a siphon without a marked trap. The left siphon contains the traps $\{\ell_4, \ell_5\}$, $\{\ell_5, \ell_6\}$, $\{\ell_6\}$, and $\{\ell_4, \ell_5, \ell_6\}$, all unmarked. The places ℓ_1 and ℓ_3 cannot be part of a trap due to the transition e_ℓ which can potentially empty the left side of the net. The union of the two siphons (i.e. the whole set of places) contains a marked trap, examples are $\{\ell_3, r_6\}$ or $\{r_3, \ell_6\}$. ■

This allows us to draw some conclusions about the existence of distinguished siphons and traps.

Lemma 2 (Maxima and Minima). *All sets in the enumeration in lemma 1 have a unique maximum (with respect to set inclusion). The set of siphons not containing a marked trap does not have a unique maximum. The only set with a unique minimum is the set of all traps.*

Proof. For those sets that are closed under union, the union of all elements is also contained in the set and therefore is the unique maximal element. The free-choice net in Fig. 1 has four different maximal siphons without a marked trap, each containing all places but two: one of ℓ_3/r_6 and one of r_3/ℓ_6. Since the set of all traps contains the

empty trap, it is minimal (but that is just a technical thing). For all other sets, take some net with a unique minimum (assuming it exists) and lay two copies of this net disjointly besides each other into one new net. This new net does not have a unique minimum anymore. ■

This tells us a lot about a good starting point when looking for a dead siphon or one that "can be made dead" as a witness against liveness: we won't find one easily. There are reasons why it may be good to make a minimal or maximal siphon (without marked trap) dead, minimal siphons are easier to grasp, maximal ones show more transitions to be dead. Since neither the minimal nor the maximal ones are unique, we may as well take one of those at random and try to empty the siphon so far that it becomes dead. The proof of theorem 1 contains a partial solution to emptying a siphon without a marked trap in free-choice nets: a circuit-free allocation α [3]. Whenever more than one transition is competing for the same token on some place s of the siphon (but not of its maximal trap, which should be empty anyway), the allocation α selects the right transition $\alpha(s)$ so that the contained traps will not become marked and the tokens are driven out of the siphon eventually. Note that either all or none of the transitions in the postset of the place s are enabled in a free-choice net, so selecting $\alpha(s)$ is always possible. The circuit-free allocation can be generalized to larger classes of nets [2] but only if the siphon does not contain any trap, which we cannot guarantee. Thus we are limited to free-choice nets. The circuit-free allocation α can easily be computed by the algorithm computeAllocation below.

Algorithm 1. computeAllocation

Require: set of places D
  $Q := D$; {start at D and continuously remove places forbidden in a trap}
  while $\exists t \in Q^\bullet$ with $Q \cap t^\bullet = \emptyset$ **do**
    for all $s \in {}^\bullet t \cap Q$ **do**
      $\alpha(s) := t$; {t leads out of the maximal trap}
      $Q := Q \backslash \{s\}$ {so it can be used to deplete s}
  return circuit-free allocation α (and maximal trap Q in D)

Consider the net from Fig. 1 again. The circuit-free allocation of the left-hand siphon is $\alpha(\ell_1) = e_\ell$, $\alpha(\ell_2) = b_\ell$, and $\alpha(\ell_3) = c_\ell$. After firing e_ℓ there are no more activated transitions in the postset of the siphon. Unluckily, the siphon is not dead either: the preset of the transition c_ℓ inside the siphon is marked, there is just an outside token on r_6 missing. We obtain a simple conclusion:

Corollary 2 (Non-local emptying). *There are free-choice nets $N = (S, T, F)$, markings m, and siphons $D \subseteq S$ not containing a marked trap (under m) such that every firing sequence leading to a marking under which D is dead contains at least one transition from $T \backslash ({}^\bullet D \cup D^\bullet)$.*

Fig. 2 shows an example of a net where no siphon can be made dead or emptied by firing just the local transitions (directly connected to it). To obtain a dead siphon in the general case we thus might require arbitrary transitions, non-local to the siphon, to fire.

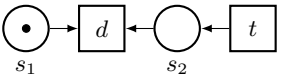


Fig. 2. The maximal (and only) siphon $\{s_1\}$ cannot be emptied (or made dead) by firing only transitions directly connected to it

On the other hand, once we have a dead siphon, we can obtain an unmarked siphon, too. To find an unmarked siphon is not harder than to find a dead one.

Lemma 3 (Every dead siphon contains an unmarked siphon). *Let D be a dead siphon at a marking m. The set D' of all places in D unmarked at m is an unmarked siphon.*

Proof. D' is obviously unmarked. Since every transition $t \in {}^\bullet D' \subseteq {}^\bullet D \subseteq D^\bullet$ has an unmarked place $s \in {}^\bullet t \cap D$ (as D is dead), $s \in D'$, $t \in D'^\bullet$ and ${}^\bullet D' \subseteq D'^\bullet$ hold. So D' is a siphon. ■

4 Partially Activated Transitions and Markable Places

We assume now that a siphon D without marked trap has been emptied as far as possible by the firing of transitions in $D^\bullet$ according to the circuit-free allocation. While there are no enabled transitions in $D^\bullet$ anymore, there still may be transitions t where ${}^\bullet t \cap D$ provides enough tokens for t, but t has some unmarked place outside D in its preset.

Definition 7 (D-enabled transitions). *Let $D \subseteq S$ be a set of places. A transition t is called* D-enabled *under m if $m(s) \geq F(s,t)$ for all $s \in D$ but $\neg m[t\rangle$.*

In Fig. 1, after firing the transition e_ℓ, the siphon $D = \{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$ has the D-enabled transition c_ℓ in its postset. It may still be possible to put enough tokens into the preset of c_ℓ to enable it, so the siphon D may not be dead. We say that the preset is markable in this case.

Definition 8 (Markable Places). *Let $s \in S$ be a place unmarked at a marking m. We call s* markable at *m if there is a marking $m' \in \mathcal{R}(m)$ with $m'(s) > 0$, otherwise we call it* unmarkable at *m.*

For free-choice nets, there is characterization of markable places [3]:

Lemma 4 (Markable Places). *Let $N = (S, T, F)$ be a free-choice net, m a marking of N, and $s \in S$ some place with $m(s) = 0$. The place s is markable at m if every siphon D with $s \in D$ contains a marked trap.*

Proof. (By contradiction.) We assume s cannot acquire a token, i.e. $m'(s) = 0$ for all $m' \in \mathcal{R}(m)$. In the following we use sets D, the siphon in construction, $dead(m')$, a set of transitions that are dead at the current marking m', and $done$, a set of transitions in the postset of D (not preventing D from being a siphon). Initially, let $D := \{s\}$, $dead(m) := {}^\bullet s$ (with the current marking being m), and $done := \emptyset$.

Choose any transition $t \in dead(m) \backslash done$. We can find a firing sequence σ such that after $m[\sigma\rangle m'$ with a new marking m', each place from ${}^\bullet t$ unmarked at m' is unmarkable at m' [3]. (Just note that t is dead, so some place in ${}^\bullet t$ must always be unmarked, and that the net is free-choice, so marked places of ${}^\bullet t$ will remain marked. All transitions consuming a token from ${}^\bullet t$ are dead, since they all have the same preset, namely ${}^\bullet t$.)

Let $s' \in {}^\bullet t$ be a place unmarkable at m'. Then, we add s' to our siphon: $D := D \cup \{s'\}$ and $dead(m') := dead(m) \cup {}^\bullet s'$. As we are done with t, we add $done := done \cup \{t\}$.

We distinguish two cases know: Either $dead(m') = done$, then for each place $s \in D$ each transition $t \in {}^\bullet s$ has a place $s' \in {}^\bullet t \cap D$ by construction, so D is a siphon. Since D is unmarked, it does not contain a marked trap, which ends the proof. Or $done$ is a proper subset of $dead(m')$, in which case we use the algorithm again, starting at the point of choosing a transition, but set the current marking to m'. Since $done$ increases its size by one for each loop and there is an finite upper limit $|T|$ on its size, at some point $dead(m'') = done$ will hold for the current marking m'' and the proof terminates. ■

Note that the free-choice property is essential in this proof, for non-free-choice nets it is possible to remove tokens from the preset of a transition t without firing t. What is possible even in free-choice nets is, of course, that a place can be markable despite being contained in a siphon without a marked trap. Consider Fig. 1 after firing e_ℓ again. The transition c_ℓ can be enabled (by putting a token on r_6) through the firing sequence $a_r d_r g_r$, which converts the siphon without marked trap $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ into a siphon with a marked trap. On the other hand, after firing e_r, the place r_6 becomes unmarkable. Removing e_r from the example net (before firing any transition) also converts R into a siphon with a marked trap, since R itself is now a trap. In that case, every siphon containing r_6 has a marked trap, thus the above lemma is applicable and r_6 can be marked. The lemma does not tell us which firing sequence to use to mark r_6, though.

What we can conclude from lemma 4 is twofold: If a place in the preset of some D-enabled transition cannot be marked, the proof of the lemma constructs a new siphon D' and a reachable marking m' with $m'(D') = 0$. This siphon is certainly dead and can be used as a witness for the non-liveness of the free-choice net. If the place can be marked, we need to calculate a firing sequence doing so. There are tools available which can do that, i.e. solve a so-called *coverability problem*. Still, we need to find out if there is a siphon containing the place in question and not having a markable trap.

5 A Siphon Interdependency Graph

We now construct a graph which tells us for each place we would like to mark if it is certainly markable, and if not, presents us a siphon without marked trap containing the place. To do that, we require some external functions we assume to be available.

Definition 9 (External Functions). *Let* Activate*:* $\mathbb{N}^S \times T \to T^*$ *be a partial function yielding a firing sequence* $\sigma \in T^*$ *with* $m[\sigma t\rangle$ *whenever such a sequence exists for a marking* $m \in \mathbb{N}^S$ *and* $t \in T$*. Let* MinSiphon*:* $S \times \mathbb{N}^S \to 2^S \cup \{\bot\}$ *yield a minimal siphon* $D \subseteq S$ *without a marked trap but with* $s \in D$ *for place* $s \in S$ *and marking*

$m \in \mathbb{N}^S$. *If this minimal siphon does not exist, the function shall yield the result* $\perp$. *Finally, let* MaxTrap*:* $2^S \to 2^S$ *yield the maximal trap* $Q \subseteq D$ *inside a given set* $D \subseteq S$.

In an implementation we used the reachability checker Sara [18,19] for the function Activate and the SAT-solver MiniSAT [13] fed with a translation of the properties for minimal siphons and maximal traps to compute the functions MinSiphon and MaxTrap. Computing a minimal siphon can be done in polynomial time in free-choice nets [23], but the function MinSiphon has to produce a minimal siphon not containing a marked trap. It may be necessary to enumerate all minimal siphons to find one without a marked trap, which can take exponential time [22]. Therefore, using MiniSAT seems a viable approach. The formulae fed to MiniSAT correspond to those discussed in [14]. Note that we will not call Activate(m, t) for pairs (m, t) for which no firing sequence exists. This is part of the guaranteed termination in the infinite state case.

Definition 10 (Siphon Interdependency Graph). *A* siphon interdependency graph *is a graph* $G = (V, v_0, \mathcal{V}, E, \mathcal{S}, \mathcal{D}, \mathcal{Q}, \alpha)$ *with the following components:*

- *a set of nodes* V *with* $|V| \leq |S| + 1$,
- *an initial node* $v_0 \in V$,
- *a map* $\mathcal{V}$*:* $S \to V$ *with* $\mathcal{V}(s) = \mathcal{V}(s') \neq v_0 \implies s = s'$,
- *a set of edges* $E \subseteq V \times T \times S \times V$, *where for each pair* (v, t) *there is at most one edge* $(v, t, s, v') \in E$, *and for every edge* $(v, t, s, v') \in E$ *holds* $v' = \mathcal{V}(s)$,
- *a map* $\mathcal{S}$*:* $V \to S \cup \{\perp\}$ *mapping each node* $v \neq v_0$ *to its unique* defining place $\mathcal{S}(v) = \mathcal{V}^{-1}(v)$; $\mathcal{S}(v_0) = \perp$ *(undefined),*
- *a map* $\mathcal{D}$*:* $V \to 2^S \cup \{\perp\}$ *attaching a siphon without a marked trap* $\mathcal{D}(v)$ *to each node* v, *with* $\mathcal{D}(v) = \perp$ *if no such siphon exists and* $\mathcal{D}(v) = \emptyset$ *as a default (uninitialized) value,*
- *a map* $\mathcal{Q}$*:* $V \to 2^S \cup \{\perp\}$ *with either* $\mathcal{Q}(v) = \perp$ *or* $\mathcal{Q}(v)$ *being the maximal trap in* $\mathcal{D}(v)$,
- *and a map* α*:* $V \to T^S \cup \{\perp\}$ *yielding for each node* $v \in V$ *either the value* $\perp$ *or a circuit-free allocation for* $\mathcal{D}(v) \backslash \mathcal{Q}(v)$.

For an edge $e = (v, t, s, v') \in E$ *let* $src(e) = v$, $trans(e) = t$, $place(e) = s$, *and* $sink(e) = v'$.

Nodes in this graph generally represent places that are contained in a siphon without a marked trap. An edge (v, t, s, v') represents the fact that a transition t in the postset of v's siphon cannot fire due to a token missing on some place s outside $\mathcal{D}(v)$. The edge then points to the node v' representing that place s, so one reason why s cannot (easily) obtain a token is the siphon $\mathcal{D}(v')$. We try to either mark $\mathcal{Q}(v')$, eliminating that reason, or make $\mathcal{D}(v')$ dead, reaching our final goal. Of course, if $\mathcal{Q}(v')$ gets marked, $\mathcal{D}(v')$ is no longer a siphon without marked trap, so the graph has to be adjusted. But let us start with creating and initializing a siphon interdependency graph G first using the algorithm createGraph. As input we expect one siphon D without a marked trap, which must exist if the net is not live. As output we obtain a graph with one (the initial) node for all the places in D and one node for each place not in D. There are no edges in the graph.

Algorithm 2. createGraph

Require: siphon D
  new graph G;
  new node v_0 **in** G;
  $\mathcal{D}(v_0) := D$; {initial node's siphon as given}
  $\mathcal{Q}(v_0) :=$ MaxTrap(D); {compute its maximal trap}
  if $m_0(\mathcal{Q}(v_0)) > 0$ **then**
    return false {if the maximal trap is marked that's an error}
  $\alpha(v_0) :=$ computeAllocation(v_0); {get the allocation}
  $\mathcal{S}(v_0) :=\perp$; {v_0 has no single defining place}
  for all $s \in D$ **do**
    $\mathcal{V}(s) := v_0$ {but all places in the siphon are defining for v_0}
  for all $s \in S \backslash D$ **do**
    new node v **in** G; {construct one node per other place}
    $\mathcal{S}(v) := s$; $\mathcal{V}(s) := v$; {which is defining for that node}
    $\mathcal{D}(v) := \emptyset$; $\mathcal{Q}(v) :=\perp$; {don't initialize anything else yet}
    $\alpha(v) :=\perp$
  return true (and the siphon interdependency graph G)

In our running example from Fig. 1 with the starting siphon $D = \{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$, the graph G initially consists of seven nodes and zero edges. Nodes are v_0 with $\mathcal{D}(v_0) = D$, $\mathcal{Q}(v_0) = \{\ell_4, \ell_5, \ell_6\}$, and v_i $(1 \leq i \leq 6)$ with $\mathcal{S}(v_i) = r_i$ and uninitialized siphons and traps.

After the construction of the graph we consider the current marking m (starting at m_0) and the siphon interdependency graph G as global objects that may be modified by our algorithms. The algorithm validateNode checks whether a node (still) represents a siphon without a marked trap at the current marking. If not, it tries to compute one and, on success, attaches it to the node. Edges starting at this node are then removed, since they represent a property of the node's siphon. In case of a failure, the node is detached from the remainder of the graph and marked as having no further interesting siphons, i.e. there are no (more) siphons without a marked trap that contain the defining place of the node. The initial node never needs to be checked; we will make sure that its siphon's maximal trap never becomes marked.

If validateNode(v_1) would be called in our example, the node's siphon would be initialized to $\mathcal{D}(v_1) = \{r_1, r_2, r_3, r_4, r_5\}$ and the maximal trap to $\mathcal{Q}(v_1) = \{r_4, r_5\}$ (no other possibilities).

If the validation of a node is successful, the algorithm will return true. Nodes with a return value of false are never used again. We need to validate a node e.g. when the firing of a transition has changed the current marking: The node represents a siphon and the siphon's maximal trap may have become marked by the firing. Thus the siphon needs to be replaced. Activated transitions are fired by the algorithm fireActivated, but only per node, i.e. only transitions lying in the postset of the currently worked upon node's siphon may be fired. If the defining place of the current node obtains a token, all edges pointing to that node become invalid. An edge represents a token demand for that place, and the demand can now be fulfilled. Transitions in "earlier" nodes may have become

Algorithm 3. validateNode

Require: node v {check if its siphon has no marked trap}
  if $v = v_0$ **then**
    return true {initial node's siphon is always ok, we won't mark its traps}
  if $\mathcal{D}(v) = \perp$ **then**
    return false {we're out of siphons without marked traps}
  if $\mathcal{D}(v) = \emptyset$ **or** $m(\mathcal{Q}(v)) > 0$ **then**
    $\mathcal{D}(v) := \mathsf{MinSiphon}(\mathcal{S}(v), m)$; {the siphon needs to be replaced}
    $\alpha(v) := \perp$; {initialize later}
    delete all edges $e \in E$ **with** $src(e) = v$; {token need has changed}
    if $\mathcal{D}(v) = \perp$ **then**
      $\mathcal{Q}(v) := \perp$; {did not find another siphon, disconnect node}
      delete all edges $e \in E$ **with** $sink(e) = v$;
      return false
    $\mathcal{Q}(v) := \mathsf{MaxTrap}(\mathcal{D}(v))$ {set the correct maximal trap}
  return true

firable, and these firings may invalidate this node's siphon. We always fire transitions from "earlier" nodes first and define the initial node as the "earliest" node. That way, it is impossible to mark the maximal trap in the initial node: Only if no transition in $\mathcal{D}(v_0)^\bullet$ is enabled other transitions are considered at all, and the circuit-free allocation $\alpha(v_0)$ ensures no token reaches the maximal trap $\mathcal{Q}(v_0)$ when a transition in $\mathcal{D}(v_0)^\bullet$ is fired.

Note that the algorithm fireActivated contains two sources of non-determinism. If there are two enabled transitions in the allocation both producing a token on the node's defining place, only one of them is fired and then fireActivated terminates. Depending on which transition gets chosen, the new marking and the future evolution of the graph may vary. The computation of the allocation itself is also non-deterministic. Additionally, the external functions MinSiphon and Activate can also introduce non-determinism. The non-determinism just reflects the fact that there may be many siphons that can become dead in many different ways. Our goal is not a specific graph here but to conclude from such a graph at least one dead siphon.

For our example, a_ℓ and e_ℓ are activated transitions in v_0. The algorithm detects the only place ℓ_1 in their preset and finds that the circuit-free allocation dictates that $\alpha(v_0)(\ell_1) = e_\ell$ must fire. We return true which will (later) guarantee that the next node to be checked is v_0 (again, in this case).

Besides the enabled transitions we also need to fire those $\mathcal{D}(v)$-enabled transitions in $\mathcal{D}(v)^\bullet$ of which we know for certain that they can be enabled. If we get a firing sequence enabling such a transition, it may happen that some transition of that firing sequence changes the marking on our initial siphon $\mathcal{D}(v_0)$. In that case, we must protect our trap $\mathcal{Q}(v_0)$ from being marked by using the algorithm activateTransition.

In our example, c_ℓ is a $\mathcal{D}(v_0)$-activated transition, but at this point we do not know if it can be activated: Its preplace r_6 lies in a siphon without a marked trap. An initialization of v_6 will lead to $\mathcal{D}(v_6) = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ and $\mathcal{Q}(v_6) = \{r_4, r_5, r_6\}$.

We finally come to the crucial part of the algorithm: adaptNode. We use the former algorithms to validate a node v and fire the activated transitions, but after that, we have

to check for $\mathcal{D}(v)$-enabled transitions and find out if they can be enabled. If none exist, our final goal is reached: the siphon $\mathcal{D}(v)$ is dead. If a $\mathcal{D}(v)$-enabled transition t can be enabled, we do it, of course. Otherwise, we find a place s outside our siphon responsible for the inactivity of t, point an edge to the according node of our graph, and check that node out. Some measures have to be taken to prohibit infinite recursion, i.e. every node may be visited at most once on a run starting at the initial node (where each run starts). We memorize the already visited nodes in a global variable $done$. The algorithm adaptNode has four possible outcomes; a result of zero means that the graph G has changed, a result of one or two means that a dead siphon has been found and we may immediately stop, and a result of three means that the subgraph starting at this node is valid and has not been changed.

Algorithm 4. fireActivated

Require: node v
  while $\exists$ enabled $t \in \mathcal{D}(v)^\bullet$ **do**
    choose $s \in {}^\bullet t \cap \mathcal{D}(v)$; {select preplace to find correct transition to fire}
    if $\alpha(v) = \perp$ **then**
      $\alpha(v) :=$ computeAllocation(v) {late initialization}
    $t := \alpha(v)(s)$; {try to empty the siphon using t}
    $m := m + t^\bullet - {}^\bullet t$; {fire t}
    print t; {document the firing sequence}
    if $\mathcal{S}(v) \in t^\bullet$ **or** ($v = v_0$ **and** $\mathcal{D}(v) \cap t^\bullet \neq \emptyset$) **then**
      {a token has been produced on the defining place of the node}
      {(for the initial node any place in the siphon will do)}
      {edges (token demands) to this place/node can be fulfilled}
      delete all edges $e \in E$ **with** $src(e) = v$ **or** $sink(e) = v$;
      return true {the graph has been modified}
  return false {the graph is unmodified}

Algorithm 5. activateTransition

Require: transition t {must be possible to enable via Lemma 4}
  $\sigma :=$ Activate(m, t); {get an enabling sequence}
  for $i := 1$ to length(σ) **do**
    if $\sigma[i] \in \mathcal{D}(v_0)^\bullet$ **then**
      {careful now: $\sigma[i]$ changes $m(\mathcal{D}(v_0))$, so check the allocation}
      choose $s \in {}^\bullet\sigma[i] \cap \mathcal{D}(v_0)$;
      $t' := \alpha(v_0)(s)$; {replace $\sigma[i]$ by t' according to the allocation}
      $m := m + t'^\bullet - {}^\bullet t'$; {fire t'}
      print t'; {document the firing sequence}
      return {break off, we changed the sequence, no continuation is possible}
    else
      $m := m + \sigma[i]^\bullet - {}^\bullet\sigma[i]$; {fire the transition}
      print $\sigma[i]$ {document the firing sequence}

Algorithm 6. adaptNode

Require: node v, set of nodes $done$ (by reference)
  $done := done \cup \{v\}$; {don't visit this node again}
  if not validateNode(v) **then**
    return 0 {could not validate the node, graph changed}
  if fireActivated(v) **then**
    return 0 {fired enabled transitions, marking changed}
  if not $\exists\, \mathcal{D}(v)$-enabled $t \in \mathcal{D}(v)^\bullet$ **then**
    print $\mathcal{D}(v)$; **return** 1 {final goal reached: a dead siphon}
  for all $\mathcal{D}(v)$-activated $t \in \mathcal{D}(v)^\bullet$ **do**
    $s := \perp$; {enabling t possible? use s as flag}
    for all $s' \in {}^\bullet t$ **with** $m(s') < F(s', t)$ **do**
      if $\mathcal{D}(\mathcal{V}(s')) = \emptyset$ **then**
        validateNode($\mathcal{V}(s')$) {siphon was not initialized so far}
      if $\mathcal{D}(\mathcal{V}(s')) \neq \perp$ **then**
        $s := s'$ {a place in a siphon without marked trap, prohibiting t}
    if $s = \perp$ **then**
      activateTransition(t); {t can be enabled, so we try to do it}
      return 0 {the graph can be invalid, possibly at an earlier node}
    else
      {check if we already know a reason s why t can't be enabled}
      if $\exists e \in E$ **with** $sink(e) = v$ **and** $trans(e) = t$ **then**
        $s := place(e)$ {edge already exists, must look at reason s}
      else $E := E \cup \{(v, t, s, \mathcal{V}(s))\}$ {new reason, new edge}
      {now recursively check the node belonging to the reason s}
      if $\mathcal{V}(s) \notin done$ **then**
        $i :=$ adaptNode($\mathcal{V}(s), done$); {visit the edge's sink node, recursion}
        if $i < 3$ **then return** i {restart or solution, make a full backtrack}
  {we completed the subgraph starting at v, no changes in the graph}
  if $v \neq v_0$ **then return** 3 {backtrack to an earlier node}
  return 2 {found the dead siphon, but it needs construction (see below)}

The only thing we have to do now is to call adaptNode for v_0 until it returns non-zero.

Consider again the running example. The first call to adaptNode for v_0 fires e_ℓ in $\mathcal{D}(v_0)^\bullet$ (as seen). The function is left for a fresh restart. On the next call, there are no more activated transitions but c_ℓ is $\mathcal{D}(v_0)$-enabled. We obtain the place $s := r_6$ as a place prohibiting the firing of c_ℓ. There are no edges in the graph yet, so a new edge is created: (v_0, c_ℓ, r_6, v_6). A recursive call to adaptNode for v_6 checks the node v_6, for which $\mathcal{D}(v_6) = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ and $\mathcal{Q}(v_6) = \{r_4, r_5, r_6\}$ are initialized. By returning zero, all previous calls to adaptNode are terminated too. We begin again with calling adaptNode for v_0, where there is still the $\mathcal{D}(v_0)$-activated transition c_ℓ, but this time, the edge already exists (and is used). Again, adaptNode is called for v_6, e_r is fired (according to the allocation computed in fireActivated(v_6)), and all calls are backtracked. On the next entry, there is no activated transition in $\mathcal{D}(v_6)^\bullet$ but c_r is $\mathcal{D}(v_6)$-enabled. Since c_r needs a token from ℓ_6 in $\mathcal{D}(v_0)$ which is a siphon without a marked trap, c_r cannot be enabled at this time. A new edge is created: (v_6, c_r, ℓ_6, v_0).

No call to adaptNode is made for v_0 though, since we have been there already. So with the final lines of the algorithm, first a value of three is returned for v_6 (we are done at this node), and then a value of two is returned for v_0, i.e. there is a dead siphon.

Note how the example would run differently if we remove e_r from the net. When trying to enable c_ℓ we find now that there is no siphon without a marked trap containing r_6. So, validateNode(v_6) yields $\mathcal{D}(v_6) = \bot$ and then removes the edge (v_0, c_ℓ, r_6, v_6) from the graph. Thus, when checking the $\mathcal{D}(v_0)$-enabled transition c_ℓ later, no place s fulfills $m(s) < F(s, c_\ell)$ as well as $\mathcal{D}(\mathcal{V}(s)) \neq \bot$, i.e., c_ℓ can be enabled according to Lemma 4. The call to activateTransition(c_ℓ) finds e.g. the firing sequence $a_r d_r g_r$ (or a longer one) which is then fired. In the next call to adaptNode for v_0, c_ℓ is now activated and will be fired. After that, $\mathcal{D}(v_0)^\bullet$ contains neither enabled nor $\mathcal{D}(v_0)$-enabled transitions, and adaptNode will print the siphon $\mathcal{D}(v_0)$ and return with a value of one.

6 Diagnosis for the Completed Graph

In the case adaptNode(v_0) returns two, no dead siphon has been printed. There is one, but we have yet to find it. The graph G now consists of a set of interlocking nodes, where each $\mathcal{D}(v)$-enabled transition of a node v needs a token on a place of some other node in the set. Since none of the nodes has any enabled transitions in its siphon's postset, the needed token cannot be delivered. Consequently, none of the $\mathcal{D}(v)$-enabled transitions can be enabled. So, all places in the siphons of all the nodes reachable from v_0 in G via edges form a dead siphon.

The algorithm deadSiphon creates the graph G, calls adaptNode as long as necessary, and if a dead siphon has not been constructed by then, collects the places of G belonging to the dead siphon.

For our example, we would collect the dead siphon $D = \mathcal{D}(v_0) \cup \mathcal{D}(v_6)$, containing all places of the net. The siphon is dead as there is only one token left (on r_3) and the only transition using that token, c_r, has another input place. Note that this siphon still contains a marked trap, $\{r_3, \ell_6\}$, but dead it is. We easily get an unmarked siphon according to Lemma 3, which consists of all places but r_3.

7 Correctness, Termination, and Experimental Results

Theorem 2 (Correctness and Termination). *The algorithm* deadSiphon *terminates with a firing sequence and a dead siphon as a counterexample to liveness when started with a siphon not containing a marked trap.*

Proof. (Sketch.) Note that there are two ways, and two ways only, for the algorithm to terminate on the input of a siphon without a marked trap. It can either exit with a return value of one after printing a dead siphon in adaptNode, or it can exit with a return value of two with a set of siphons without activated transitions (in the postsets). In this set every $\mathcal{D}(v)$-enabled transition for some node v needs an additional token from one of the other siphons, which it cannot get. Therefore the union over the set of siphons is a dead siphon.

Algorithm 7. deadSiphon

Require: siphon D
  global marking $m := m_0$; {marking globally available}
  if not createGraph(D) **then**
    return error {the given siphon has a marked trap}
  {the graph G is now globally available}
  $i := 0$; {enter the construction loop at least once}
  while $i = 0$ **do**
    $done := \emptyset$; {for avoiding loops via adaptNode}
    $i :=$ adaptNode(v_0,$done$);
  if $i \neq 2$ **then**
    return {firing sequence and dead siphon have already been printed}
  $D' := \mathcal{D}(v_0)$; {construct a dead siphon by collecting all the places reachable}
  $A := \{v_0\}$; $C := \emptyset$; {all nodes reachable / completely visited nodes}
  while $A \neq C$ **do**
    choose $v \in A \backslash C$; {take a non-visited node v}
    $C := C \cup \{v\}$; {mark it as visited}
    for all $e \in E$ **with** $src(e) = v$ **do**
      $D' := D' \cup \mathcal{D}(sink(e))$; {for each edge from v, add the sink's siphon}
      $A := A \cup \{sink(e)\}$ {and mark the sink as reachable}
  print D'; {print the dead siphon and terminate}

The only other function in the algorithm that could make it halt in any way is Activate, as it is a partial function. Since we know by Lemma 4 that a transition t can be enabled from m before calling Activate(t,m), the error can never occur. Thus, upon termination, a witness has been found.

For termination we study a progress condition. Every time adaptNode(v_0,$done$) returns with value zero in deadSiphon, one of three things has happened: At least one transition has fired, or a node's siphon has been replaced, or a $\mathcal{D}(v)$-enabled transition (for some v) has been activated. In the latter case a transition has been fired, too. The second case can happen at most as often as there are siphons without marked traps (plus a constant for setting the node to "no more siphons") times $|S|$ (since many nodes can point to the same siphon), which is a finite number for any net.

When a transition is fired, this is done according to a circuit-free allocation, such that the siphon becomes emptier (counting e.g. weighted tokens, each place s appearing in an allocation having as weight factor $w(s) := 1 + \max_{t \in s^\bullet} \sum_{s' \in t^\bullet} w(s')$, all other places having weight zero). Unless a siphon becomes invalid and is replaced, all siphons in the graph can only become emptier. Note now, that by design the initial node's siphon never gets a marked trap, so at least one siphon must always remain in the graph. Assuming non-termination, we thus reach a point when all (at least one) siphons in the graph have their minimal weight, i.e. they are dead, no more siphons are available as replacement, and the algorithm continues running. This is a contradiction, as the algorithm terminates immediately when the initial node's siphon (or any other it works on) becomes dead. ■

The algorithm obviously needs more than polynomial-time in the worst case due to the non-polynomial runtime of the external functions Activate and MinSiphon and the

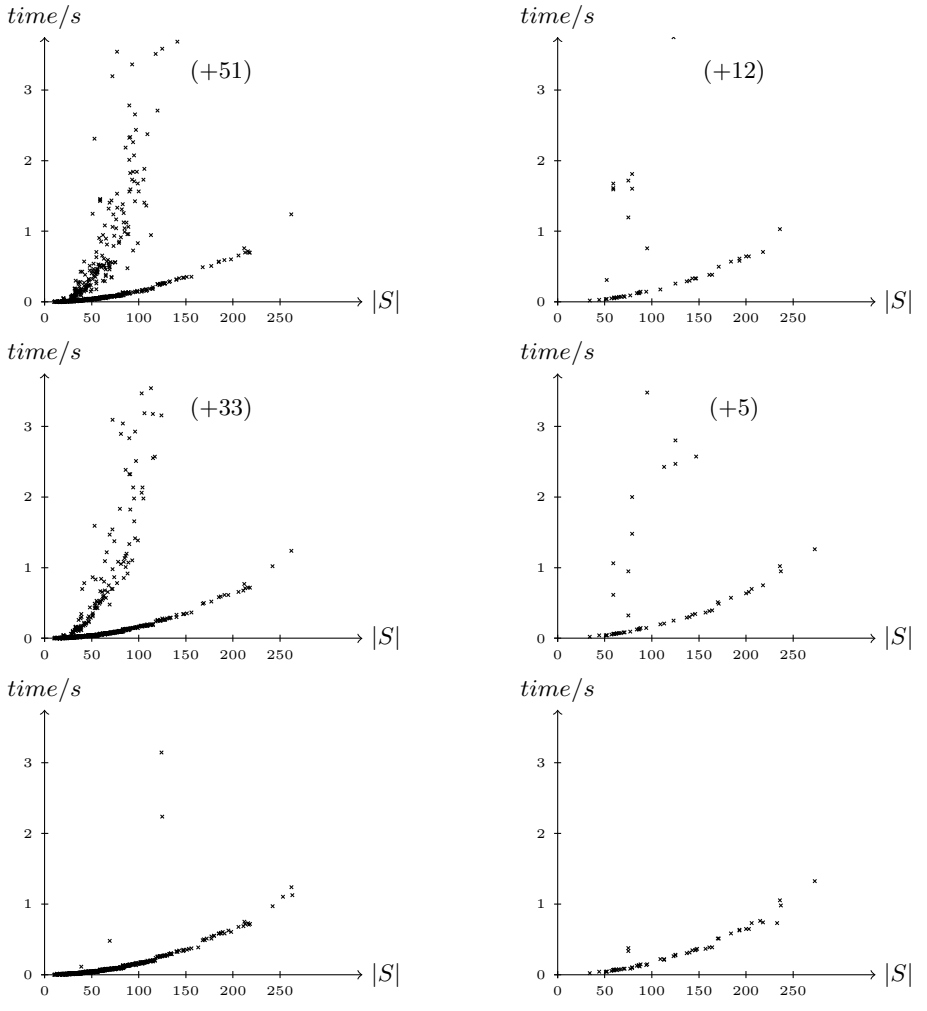


Fig. 3. The figures show the relation between the total number of places in a net and the time needed to compute a witness path against liveness. Checking liveness and computing the initial siphon are not included in these times. Inscriptions $(+x)$ denote x nets taking too much time to be shown in the figure. Figures on the left contain only bounded nets (737 nets per figure), figures on the right those nets that were either unbounded or where LoLA could not make a decision (59 npf). The three rows distinguish between the initially used siphons, top: minimal siphons, middle: arbitrary siphons (whatever MiniSAT yielded), bottom: maximal siphons.

number of siphons being explored, while all other parts of the algorithm only contribute polynomial factors. Especially reducing the number of siphons to be explored could therefore be a good way to reduce the average case complexity of the algorithm and make it applicable in most cases.

We have implemented the algorithm and tested it with a set of 1386 business processes with sizes from four to 273 places, 47 on average. All nets were free-choice and had one input and one output place which are connected (from output to input) by a new transition, making the liveness check equivalent to a check for weak termination, i.e. whether a token can reach the output place from every reachable marking. The results were produced on a 2.6GHz PC with 4GB RAM under Windows/Cygwin and compared to results of the state space analyzer LoLA [21].

Our first experiment included the test for liveness and production of a siphon without a marked trap as input to the algorithm. 590 of the 1386 nets were found live, so no witnesses were produced. Our implementation succeeded to present a witness in the 796 remaining cases, using 21 minutes for all 1386 nets together, split into about 5 minutes for the liveness check and 16 minutes for the witness path construction.

We then used the state space exploration tool LoLA to obtain a rough comparison for the liveness test. LoLA comes in two different configurations: one for safe nets only and one for arbitrary nets. The safe configuration took only 40 seconds to run, but for 385 nets the computation was stopped immediately when an unsafe marking was encountered, liveness not having been decided. In the arbitrary configuration LoLA took 28 minutes and still 47 nets remained undecided due to a lack of memory. 46 of these 47 nets are infinite state systems. An exact quantification was difficult, but the impression was that LoLA spent more than 80% of the time for the 47 nets.

For further experiments the 796 non-live nets were separated into 737 bounded nets and 59 nets where the state space is either infinite or so large that LoLA could not decide. For each net the time needed to compute the witness path was measured (but without the liveness test and the construction of an initial siphon not containing a marked trap). Fig. 3 shows three experiments (from top to bottom, bounded nets on the left, unbounded on the right) where for each net the computation time is measured against the size (number of places). In general, we see two branches in each picture, one that looks like a quadratic polynomial and one that goes through the roof and is clearly exponential. The only difference between the three experiments is the choice of the initial siphon: in the top row we use minimal siphons, in the middle the arbitrary siphons constructed by MiniSAT, and in the bottom row maximal siphons. This suggests that the performance of the witness path construction improves with the size of the initial siphon and that the polynomial-time reduction of its size that can be achieved during a liveness check [1] is rather detrimental.

8 Conclusion

We have shown an algorithm that turns a diagnosis information of type “there is a siphon without included marked trap” into a diagnosis information “if you fire this sequence, the following transitions mutually block each other”. The latter information is obviously easier to comprehend by non-experts. In addition, it is more similar to diagnostic information provided by state space methods. This way, we presented a step forward to a better applicability of structural techniques off the shelf. We provided experimental

results that suggest that our method is competitive to state space methods. Moreover, both the investigation of siphons and traps as such *and* our method for computing diagnostic information are guaranteed to terminate even for unbounded free-choice nets, adding a clear improvement to existing state space methods.

References

1. Barkaoui, K., Minoux, M.: A Polynomial-Time Graph Algorithm to Decide Liveness of Some Basic Classes of Bounded Petri Nets. In: Jensen, K. (ed.) ICATPN 1992. LNCS, vol. 616, pp. 62–75. Springer, Heidelberg (1992)
2. Barkaoui, K., Pradat-Peyre, J.: On Liveness and Controlled Siphons in Petri Nets. In: Billington, J., Reisig, W. (eds.) ICATPN 1996. LNCS, vol. 1091, pp. 57–72. Springer, Heidelberg (1996)
3. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
4. D'Souza, K., Khator, S.: A survey of Petri net applications in modeling controls for automated manufacturing systems. Computers in Industry 24(1), 5–16 (1994)
5. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, Springer, Heidelberg (2009)
6. Hack, M.: Analysis of Production Schemata by Petri Nets. Massachusetts Institute of Technology, 1972. MS Thesis, corrected (1974)
7. Hack, M.: Decidability Questions for Petri Nets. Massachusetts Institute of Technology, Ph.D. Thesis (1976)
8. Jones, N., Landweber, L., Lien, Y.: Complexity of some problems in petri nets. Theoretical Computer Science 4, 277–299 (1977)
9. Kosaraju, S.R.: Decidability of reachability in vector addition systems. In: Proceedings of the 14th Annual ACM STOC, pp. 267–281 (1982)
10. Lambert, J.L.: A structure to decide reachability in Petri nets. Theoretical Computer Science 99, 79–104 (1992)
11. Lipton, R.J.: The Reachability Problem Requires Exponential Space. Research Report, 62 (1976)
12. Mayr, E.: An algorithm for the general Petri net reachability problem. SIAM Journal of Computing 13(3), 441–460 (1984)
13. MiniSat. Minimalistic, open-source SAT solver (2007), `http://www.minisat.se`
14. Oanea, O., Wimmel, H., Wolf, K.: New Algorithms for Deciding the Siphon-Trap Property. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 267–286. Springer, Heidelberg (2010)
15. Talcott, C., Dill, D.: The pathway logic assistent. In: Third International Workshop on Computational Methods in Systems Biology (2005)
16. van der Aalst, W.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
17. Wimmel, H.: Entscheidbarkeit bei Petri Netzen. eXamen.press. Springer, Habilitation Thesis (2008)
18. Wimmel, H.: Sara – Structures for Automated Reachability Analysis (2010), `http://service-technology.org/tools/download`
19. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri Net State Equation. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, Springer, Heidelberg (2011)

20. Wolf, K.: Generating Petri net state spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)
21. Wolf, K.: LoLA – A low level analyzer (2010), http://www.informatik.uni-rostock.de/~nl/wiki/tools/lola
22. Yamauchi, M., Watanabe, T.: Algorithms for Extracting Minimal Siphons Containing Specified Places in a General Petri Net. IEICE Trans. Fundamentals E82-A(11), 2566–2572 (1999)
23. Yamauchi, M., Watanabe, T.: Time Complexity Analysis of the Minimal Siphon Extraction Problem of Petri Nets. IEICE Trans. Fundamentals E82-A(11), 2558–2565 (1999)

A Petri Net Interpretation of Open Reconfigurable Systems*

Frédéric Peschanski[1], Hanna Klaudel[2], and Raymond Devillers[3]

[1] UPMC – LIP6
[2] Université d'Évry–IBISC
[3] Université Libre de Bruxelles

Abstract. We present a Petri net interpretation of the pi-graphs - a graphical variant of the pi-calculus. Characterizing labelled transition systems, the translation can be used to reason in Petri net terms about open reconfigurable systems. We demonstrate that the pi-graphs and their translated Petri nets agree at the semantic level. In consequence, existing results on pi-graphs naturally extend to the translated Petri nets, most notably a guarantee of finiteness by construction.

Keywords: Reconfigurable systems, Pi-calculus, Petri nets.

1 Introduction

Systems that reconfigure themselves dynamically are ubiquitous: from Internet servers that create and destroy dynamic connections with clients, to mobile systems that scrutinize their surrounding environment to adapt their behaviour dynamically. The pi-calculus [1] is acknowledged as a particularly concise and expressive formalism to specify the dynamic structure of reconfigurable systems.

There is now a relatively rich tradition of translations of pi-calculus variants into Petri nets. One motivation of such studies is the possibility to apply efficient verification techniques on pi-calculus models. Another interest relates to expressivity: which features of the pi-calculus can be translated? What is the abstraction level required from the target Petri net formalism? Existing related research studies roughly decompose in either semantic [2,3,4] or syntax-driven [5] translations. In the first case, a reachability analysis of the semantics of a given term is performed, and a Petri net is synthesized from this. In the second case, a Petri net is built directly from the syntax. Most of the time, the semantic analysis allows to produce lower level nets. On the other side of the coin, semantic encodings generally cover less features (especially no support for match or mismatch operators; *i.e.*, without comparison of names) and often only apply on *reduction*, or *chemical* semantics for *closed systems*. With the increased expressivity of high-level nets, it is possible to support more constructs. Most

* This research is supported by National Natural Science Foundation of China under Grant No. 60910004.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 208–227, 2011.

importantly, it is possible to capture the semantics of *open systems* by translating the richer *labelled transition semantics* which is most customary in studies about the pi-calculus. To quote Robin Milner: *"you can't do behavioural analysis with the chemical semantics [...] I think the strength of labels is that you get the chance of congruential behaviours"* [6]. More prosaically, labelled transition semantics enables compositional reasoning about partial specifications.

In this paper, we continue our work about translating rich pi-calculus variants in labelled transition semantics [5]. By introducing a "translation-friendly" variant of the pi-calculus, namely the pi-graphs [7,8], we are able to provide a much simpler syntax-driven translation to one-safe coloured Petri nets. The translation supports most of the constructs of the original pi-calculus, including the match and mismatch operators with their non-trivial semantics. For non-terminating behaviours we use a notion of *iterator* along the line of [9]. The semantics for the intermediate calculus rely on graph relabelling techniques (hence the name pi-graphs) but in the present paper, to avoid confusion with the Petri net semantics, we provide a more abstract presentation. An important result of the paper is that these semantics are in *agreement*. In consequence, existing results on pi-graphs (cf. [8]) naturally extend to the translated Petri nets, most notably a guarantee that only finite-state systems can be constructed.

2 The Process Calculus

The pi-graph process calculus is a variant of the pi-calculus that enjoys a natural graphical interpretation[1] hence its name. The syntax is based on prefixes, processes, iterators and graphs. The prefixes are as follows :

$$p ::= \tau \mid \overline{c}\langle a\rangle \mid c(x) \mid \sum[P_1 + \ldots + P_n] \mid \prod[P_1 \parallel \ldots \parallel P_n] \quad (n > 1)$$

The first three elements correspond to the standard action prefixes of the pi-calculus: silent action τ, output $\overline{c}\langle a\rangle$ of name a on channel c, and input $c(x)$ through channel c of a name bound to the variable x. The remaining two elements are an n-ary non-deterministic choice operator $\sum$ between a set of possible processes $P_1, \ldots, P_n$ and an n-ary parallel operator $\prod$. Unlike most variants of the pi-calculus, these are not considered as top-level binary operators. The reason is that the sub-processes of choice and parallel must be *terminated* (cf. below). The syntax for processes is as follows:

$$P, Q, P_i ::= p.P \mid [a = b]P \mid [a \neq b]\ P \mid p.0$$

A process can be constructed inductively by prefixing another process P by a prefix p, denoted $p.P$. The match $[a = b]\ P$ (resp. mismatch $[a \neq b]\ P$) is such that the process P is only enabled if the names a and b are proved equal (resp. inequal). Finally, the construction $p.0$ corresponds to the suffixing of a prefix p

[1] The graphical interpretation of pi-graphs is not detailed in the present paper to avoid any confusion with the translated Petri nets. This interpretation is discussed at length in previous publications [7,8].

by a *terminator* 0. It is the only way to terminate a prefixed sequence. Moreover, no match or mismatch is allowed in front of 0, for semantic reasons (cf. [8]).

The top-level components of pi-graphs are iterators, which allow to encode control-finite recursive behaviours without relying on explicit recursive calls. The syntax of iterators is as follows:

$\mathcal{I} ::= I : (\nu a_1, \ldots, \nu a_n) \ * P$ with $a_1, \ldots, a_n$ names, P a process and I a label

The behaviour of an iterator is the repeated execution of its main process P in which the names $a_1, \ldots, a_n$ are considered *locally private*.

Iterators can be composed to form a pi-graph with the following syntax:

$\pi ::= (\nu A_1, \ldots, \nu A_m)\ \mathcal{I}_1 \parallel \ldots \parallel \mathcal{I}_p$ with $A_1, \ldots, A_m$ names and $\mathcal{I}_1, \ldots, \mathcal{I}_p$ iterators

The names $A_1, \ldots, A_m$ are considered *globally private* in all the iterators $\mathcal{I}_1, \ldots, \mathcal{I}_p$. These (roughly) play the role of static restrictions as in CCS, whereas locally private names (i.e. $\nu a_1, \ldots, \nu a_n$) correspond to the (dynamic) restrictions of the pi-calculus. Iterators are implicitly executed in parallel.

To illustrate the syntax constructors and their informal meaning, we consider the example of a simple server model.

Example 1. a sequential server

$$(\nu \mathrm{Start}, \nu \mathrm{Run}, \nu \mathrm{End})$$

$$\mathrm{Service} : *\mathrm{Run}(s).s(m).\sum \left[\begin{array}{l} [m = s]\ \overline{\mathrm{End}}\langle s \rangle.0 \\ +\ [m \neq s]\ \tau.\overline{\mathrm{Start}}\langle s \rangle.0 \end{array} \right].0 \quad \parallel$$

$$\mathrm{Handler} : *\mathrm{Start}(s).\overline{\mathrm{Run}}\langle s \rangle.0 \quad \parallel$$

$$\mathrm{Server} : (\nu \mathrm{session})\ * \mathrm{adr}(c).\overline{c}\langle \mathrm{session} \rangle.\overline{\mathrm{Start}}\langle \mathrm{session} \rangle.\mathrm{End}(q).0$$

The depicted pi-graph has three iterators: the main server process, a session handler and a service delivered to clients. The server initially expects a connection request from a client on the public channel adr. A (locally) private session channel is then sent back to the connected client and an instance of the service is started through the session handler. The (globally) private channels - Start, Run and End - are used to confine the server model, disallowing external interferences. When a request is emitted by a client (through channel s in the Service iterator) two cases are discriminated. If the received message m is the session channel s itself, then this indicates the end of the session. The control is passed back to the Server iterator using the (globally private) End channel. Otherwise, an internal activity is recorded (as a silent step τ) and the service is reactivated by the Handler component.

Note that in this example we only specify the server infrastructure. We see the model as an open system independent from any particular client model.

3 Operational Semantics

The operational semantics of pi-graphs is based on *graph relabelling*, a simplistic form of *graph rewriting* where nodes and edges can have their content updated

but neither created nor deleted. To avoid any confusion with the Petri net semantics - also a form of graph relabelling - we adopt in this paper a process calculus presentation. As a preliminary, we require a precise definition of what is a name in a pi-graph.

Definition 1. *The set of* ***names*** $\mathcal{N}$ *is the disjoint union of the following sets:*

$\mathcal{N}_f$ *the set of free names* $a, b, \ldots$
$\mathcal{N}_v$ *the set of variables* $x^I, y^J, \ldots$ *with* I, J *iterator labels*
$\mathcal{N}_r$ *the set of restrictions* $\nu A, \nu B, \ldots$
$\mathcal{N}_p$ *the set of private names* $\nu^I a, \nu^J b, \ldots$ *with* I, J *iterator labels*
$\mathcal{N}_s$ *the set of shared names* $\nu^I a_{:k}, \nu^I b_{:k'} \ldots$ $(k, k' \in \mathbb{N})$
$\mathcal{N}_o \stackrel{\text{def}}{=} \{n! \mid n \in \mathbb{N}\}$ *the set of output names*
$\mathcal{N}_i \stackrel{\text{def}}{=} \{n? \mid n \in \mathbb{N}\}$ *the set of input names*

We define $\mathsf{Priv} \stackrel{\text{def}}{=} \mathcal{N}_r \cup \mathcal{N}_p \cup \mathcal{N}_s$ *(private names) and* $\mathsf{Pub} \stackrel{\text{def}}{=} \mathcal{N} \setminus \mathsf{Priv}$ *(public names)*

This categorization is required because names are *globalized* in the semantics (i.e. all names have global scope). The sets $\mathcal{N}_i$ and $\mathcal{N}_o$ are names whose identity is generated fresh by construction. They play a prominent role in the model.

3.1 Terms, Context and Initial State

The semantics manipulates terms of the form $\Gamma \vdash \pi$ where π is a pi-graph with names globalized (cf. below) and $\Gamma = \beta; \gamma; \kappa$ is a global context with:

- $\beta \in \mathcal{N} \to \mathcal{N}$ a name instantiation function,
- $\kappa \in \mathcal{N}_o \to \mathbb{P}(\mathcal{N}_i)$ a causal clock, and
- $\gamma \stackrel{\text{def}}{=} (\gamma^=, \gamma^{\neq}) \in (\mathcal{N} \times \mathcal{N})^2$ a dynamic match/mismatch partition

The operations available on the context are formalized in Table 1. We will discuss these in the remainder of the section.

The initial state and context is denoted $\beta_0; \gamma_0; \kappa_0 \vdash \pi_0$ where π_0 is a syntactic term with globalized names. If the syntactic term is $(\nu A_1, \ldots, \nu A_m)\ \mathcal{I}_1 \parallel \ldots \parallel \mathcal{I}_p$ then we transform it as $(\mathcal{I}_1 \parallel \ldots \parallel \mathcal{I}_p)\{\nu A_1/A_1, \ldots, \nu A_m/A_m\}$, i.e. each restricted bound name A_i is replaced by an explicit restricted occurrence νA_i in the set $\mathcal{N}_r$. Moreover, we rewrite each iterator of the form $I_j : (\nu a_1, \ldots, \nu a_n) * P_j$ as $I_j : *P_j\{\nu^{I_j} a_1/a_1, \ldots, \nu^{I_j} a_n/a_n\}$ where each name $\nu^{I_j} a_k$ is in $\mathcal{N}_p$. As such, we encode at the global level the local nature of these private names. Finally we repeat this process for each input variable x of iterator I_j, renamed (as well as its occurrences) as x^{I_j} in set $\mathcal{N}_v$. We thus end up with a term containing only names with global scope but ranging from disjoint subsets of $\mathcal{N}$ (all iterator labels are assumed different).

In the initial state, the instantiation function associates each name of π_0 with itself, i.e. $\beta_0 \stackrel{\text{def}}{=} \{n \mapsto n \mid n \text{ a name of } \pi_0\}$. By default, there are no explicit name (in)equality so the partitions are empty, i.e. $\gamma_0 \stackrel{\text{def}}{=} (\emptyset, \emptyset)$. Finally, the initial clock is also empty, i.e. $\kappa_0 \stackrel{\text{def}}{=} \emptyset$. The initial state of the server example (cf. Example 1) is depicted below.

Table 1. The global context and associated operations

Instantiation β	
reference	$\beta(x) \stackrel{\text{def}}{=} \beta(x)$ if $x \in \mathsf{dom}(\beta)$, x otherwise
update[2]	$\beta[y/x] \stackrel{\text{def}}{=} \beta \rhd \{x \mapsto y\} \cup \{z \mapsto y \mid \beta(z) = x\}$
reset	$\mathsf{reset}^I(\beta) \stackrel{\text{def}}{=} (\beta \setminus \{\nu^I a \mapsto y \mid \nu^I a \in \mathsf{dom}(\beta)\})$
	$\rhd\{z \mapsto \nu^I a_{:k} \mid z \mapsto \nu^I a \in \beta,\ k = \mathsf{min}(\mathbb{N}^+ \setminus \{k' \mid \nu^J b_{:k'} \in \mathsf{codom}(\beta)\})\}$
Causal clock κ	
fresh output	$\mathsf{out}(\kappa) \stackrel{\text{def}}{=} \kappa \cup \{\mathsf{next_o}(\kappa)! \mapsto \emptyset\}$
fresh input	$\mathsf{in}(\kappa) \stackrel{\text{def}}{=} \{o \mapsto (\kappa(o) \cup \{\mathsf{next_i}(\kappa)?\}) \mid o \in \mathsf{dom}(\kappa)\}$
freshness (in)	$\mathsf{next_i}(\kappa) \stackrel{\text{def}}{=} \mathsf{min}\left(\mathbb{N}^+ \setminus \{n \mid n? \in \bigcup \mathsf{cod}(\kappa)\}\right)$
freshness (out)	$\mathsf{next_o}(\kappa) \stackrel{\text{def}}{=} \mathsf{min}\left(\mathbb{N}^+ \setminus \{n \mid n! \in \mathsf{dom}(\kappa)\}\right)$
r-w causality	$x \prec_\kappa y \stackrel{\text{def}}{=} x \in \mathsf{dom}(\kappa) \wedge y \in \kappa(x)$
unify	$\kappa_{\lhd x=y} \stackrel{\text{def}}{=} \bigcup_{n! \in \mathsf{dom}(\kappa)} n! \mapsto E$ s.t. $\begin{cases} E = \kappa(n!) \setminus \{x\} \text{ if } y \notin \kappa(n!) \\ E = \kappa(n!) \setminus \{y\} \text{ if } x \notin \kappa(n!) \\ E = \kappa(n!) \text{ otherwise} \end{cases}$
Partitions	$\gamma \stackrel{\text{def}}{=} (\gamma^=, \gamma^\neq)$
match	$x =_\gamma y$ iff $\begin{cases} x = y \vee (x,y) \in \gamma^= \vee (y,x) \in \gamma^= \\ \vee \exists z,\ x =_\gamma z \wedge z =_\gamma y \end{cases}$
mismatch	$x \neq^\kappa_\gamma y$ iff $\begin{cases} (x,y) \in \gamma^\neq \vee (y,x) \in \gamma^\neq \vee (x, y \in \mathsf{Priv} \cup \mathcal{N}_o \wedge x \neq y) \\ \vee (x \in \mathcal{N}_f \wedge y \in \mathsf{Priv} \cup \mathcal{N}_o) \vee (y \in \mathcal{N}_f \wedge x \in \mathsf{Priv} \cup \mathcal{N}_o) \\ \vee (x \in \mathcal{N}_o \wedge y \in \mathcal{N}_i \wedge x \not\prec_\kappa y) \\ \vee (y \in \mathcal{N}_o \wedge x \in \mathcal{N}_i \wedge y \not\prec_\kappa x) \\ \vee (\exists z_1\ z_2,\ x =_\gamma z_1 \wedge z_1 \neq^\kappa_\gamma z_2 \wedge z_2 =_\gamma y) \end{cases}$
compatibility	$x \overset{\gamma}{\sim}_\kappa y$ iff $\begin{cases} \neg(x \neq^\kappa_\gamma y) \wedge (x, y \in \mathcal{N}_f \cup \mathcal{N}_i \\ \vee (x \in \mathcal{N}_o \wedge y \in \mathcal{N}_i \wedge x \prec_\kappa y) \vee x =_\gamma y \vee y \overset{\gamma}{\sim}_\kappa x) \end{cases}$
refine	$\gamma_{\lhd x=y} \stackrel{\text{def}}{=} (\gamma^= \cup \{(x,y)\}, \gamma^\neq)$ if $\neg(x =_\gamma y)$, γ otherwise
refine set	$\gamma \lhd \{(x,y)\} \cup E \stackrel{\text{def}}{=} \gamma_{\lhd x=y} \lhd E$ and $\gamma \lhd \emptyset \stackrel{\text{def}}{=} \gamma$
discriminate	$\gamma_{\lhd x \neq^\kappa y} \stackrel{\text{def}}{=} (\gamma^=, \gamma^\neq \cup \{(x,y)\})$ if $\neg(x \neq^\kappa_\gamma y)$, γ otherwise

Example 2. the sequential server with global (and shortened) names

$\beta_0; \gamma_0; \kappa_0 \vdash$

$$V : \boxed{*}\ \nu\mathrm{Run}(x^V).x^V(y^V).\sum \begin{bmatrix} [y^V = x^V]\ \overline{\nu\mathrm{End}}\langle x^V\rangle.0 \\ +\ [y^V \neq x^V]\ \tau.\overline{\nu\mathrm{Start}}\langle x^V\rangle.0 \end{bmatrix}.0 \quad \|$$

$$H : \boxed{*}\ \nu\mathrm{Start}(x^H).\overline{\nu\mathrm{Run}}\langle x^H\rangle.0 \quad \|$$

$$S : \boxed{*}\ \mathrm{a}(x^S).\overline{x^S}\langle \nu^S\mathrm{ses}\rangle.\overline{\nu\mathrm{Start}}\langle \nu^S\mathrm{ses}\rangle.\nu\mathrm{End}(q^S).0$$

To model control-flow we use boxes to surround the active parts of pi-graphs. A prefix denoted $\boxed{p}$ is considered active, and it is said to be a *redex*. Anticipating on the Petri net semantics, the redex prefixes will be associated to places with an active control token inside. The left-part of a match $[a = b]\ P$ (resp. mismatch $[a \neq b]\ P$) can also be a redex, which is denoted $\boxed{[a = b]}P$ (resp. $\boxed{[a \neq b]}P$).

[2] For partial functions f_1 and f_2, $f_1 \rhd f_2 \stackrel{\text{def}}{=} f_1 \setminus \{x \mapsto f_1(x) \mid x \in \mathsf{dom}(f_2)\} \cup f_2$.

The 0 suffix of a process can also be a redex $\boxed{0}$, which means the process has terminated. Finally, an iterator $I : *P$ can be a redex, denoted $I : \boxed{*}P$. This means the iterator is ready to execute a (first or) new iteration of P.

A process in its initial state is denoted $\boxed{P}$ such that:

$\boxed{p.P} \stackrel{\text{def}}{=} \boxed{p}.P$, $\boxed{p.0} \stackrel{\text{def}}{=} \boxed{p}.0$ and $\boxed{[a=b]P} \stackrel{\text{def}}{=} \boxed{[a=b\,]}\ P$ (resp. $\neq$).

A process is thus in its initial state if its initial prefix is a redex. We also introduce a complementary notation for a process in its terminal state, which we denote $\boxed{\boxed{P}}$, when the terminator of a process is a redex, i.e:

$\boxed{\boxed{p.P}} \stackrel{\text{def}}{=} p.\boxed{\boxed{P}}$, $\boxed{\boxed{p.0}} \stackrel{\text{def}}{=} p.\boxed{0}$ and $\boxed{\boxed{[a=b]P}} \stackrel{\text{def}}{=} [a=b]\boxed{\boxed{P}}$ (resp. $\neq$).

In the initial state, the term π_0 has all and only its iterators marked as redex, i.e. π_0 is of the form $I_1 : \boxed{*}P_1 \parallel \ldots \parallel I_p : \boxed{*}P_p$, cf. Example 2 above.

3.2 The Semantic Rules

The operational semantics rules are listed in Table 2. Each rule is of the form: $\beta;\gamma;\kappa \vdash \theta \xrightarrow{\alpha} \beta';\gamma';\kappa' \vdash \theta'$ where θ is a subterm of a pi-graph π with some redex(es) inside[3]. We write $\pi[\theta]$ to denote π with the designated subterm θ. The local application of a rule to the subterm θ is reflected globally as follows:

Definition 2. *A* ***global transition*** $\beta;\gamma;\kappa \vdash \pi[\theta] \xrightarrow{\alpha} \mathsf{gc}(\beta';\gamma';\kappa') \vdash \pi[\theta']$ *is inferred iff* $\beta;\gamma;\kappa \vdash \theta \xrightarrow{\alpha} \beta';\gamma';\kappa' \vdash \theta'$ *is provable*

In a global transition, the label α can be either a low-level rewrite ϵ, a silent step τ, an output $\overline{c}\langle a\rangle$ or an input $c(x)$. In the subterm θ' after the rewrite, the only possibility is a move of redex(es): i.e. an evolution of the control graph. Moreover, the rules are *local*, no other subterm of π is changed. The global context, however, can be updated through a rewrite. Also, the gc function "garbage collects" all the *inactive* names from the context. This clean-up is required to ensure the finiteness of the model (cf. [8]).

Definition 3. *In a global context* $\beta;\gamma;\kappa$*, the set of* ***inactive names*** *is:*

$$\mathsf{inact}(\beta;\gamma;\kappa) \stackrel{\text{def}}{=} \{n \mid (n \in \mathcal{N}_i \cup \mathcal{N}_s \wedge n \notin \mathsf{cod}(\beta)) \vee (n \in \mathcal{N}_o \wedge n \notin \mathsf{cod}(\beta) \wedge \neg(\exists m \in \mathsf{cod}(\beta),\ n =_\gamma m))\}$$

All the static names, i.e. the names in $\mathcal{N}\backslash(\mathcal{N}_i \cup \mathcal{N}_o \cup \mathcal{N}_s)$ are considered active. An input or a shared name n is considered inactive if and only if it is not instantiated in β. For an output name the situation is slightly more complicated because it is possible that the name has been equated to another name in γ, which means that even if it is not instantiated its existence may be required. For example, if two names $o!$ and $i?$ are considered equal (i.e $o! =_\gamma i?$), even if $o!$ is not instantiated (i.e. $o! \notin \mathsf{cod}(\beta)$), any other $o'!$ must be considered distinct from $i?$ as long as the latter remains itself instantiated.

[3] The shape of a subterm θ in a pi-graph $\pi[\theta]$ follows the left-hand side of the semantic rules (cf. Table 2). The only non-trivial case is for the [sync] rule because the subterm involves two separate sub-processes, potentially in distinct iterators. In [8] such a subterm indeed corresponds to a subgraph.

Table 2. The operational semantics rules

$$[\text{silent}]\quad \beta;\gamma;\kappa \vdash \boxed{\tau}.P \xrightarrow{\tau} \beta;\gamma;\kappa \vdash \tau.\boxed{P}$$

$$[\text{out}]\quad \beta;\gamma;\kappa \vdash \boxed{\overline{\Phi}\langle\Delta\rangle}.P \xrightarrow{\overline{\beta(\Phi)}\langle\beta(\Delta)\rangle} \beta;\gamma;\kappa \vdash \overline{\Phi}\langle\Delta\rangle.\boxed{P} \quad \text{if } \beta(\Phi),\beta(\Delta) \in \mathsf{Pub}$$

$$[\text{o-fresh}]\quad \beta;\gamma;\kappa \vdash \boxed{\overline{\Phi}\langle\Delta\rangle}.P \xrightarrow{\overline{\beta(\Phi)}\langle \mathsf{next_o}(\kappa)! \rangle} \beta[\mathsf{next_o}(\kappa)/\Delta];\gamma;\mathsf{out}(\kappa) \vdash \overline{\Phi}\langle\Delta\rangle.\boxed{P}$$
$$\text{if } \beta(\Phi) \in \mathsf{Pub},\ \beta(\Delta) \in \mathsf{Priv}$$

$$[\text{i-fresh}]\quad \beta;\gamma;\kappa \vdash \boxed{\Phi(x)}.P \xrightarrow{\beta(\Phi)(\mathsf{next_i}(\kappa)?)} \beta[\mathsf{next_i}(\kappa)?/x];\gamma;\mathsf{in}(\kappa) \vdash \Phi(x).\boxed{P}$$
$$\text{if } \beta(\Phi) \in \mathsf{Pub}$$

$$[\text{match}]\quad \beta;\gamma;\kappa \vdash \boxed{[\Phi = \Delta]}P \xrightarrow{\varepsilon} \beta;\gamma_{\lhd\beta(\Phi)=\beta(\Delta)};\kappa_{\lhd\beta(\Phi)=\beta(\Delta)} \vdash [\Phi = \Delta]\boxed{P}$$
$$\text{if } \beta(\Phi) \sim_\kappa \beta(\Delta)$$

$$[\text{miss}]\quad \beta;\gamma;\kappa \vdash \boxed{[\Phi \neq \Delta]}P \xrightarrow{\varepsilon} \beta;\gamma_{\lhd\beta(\Phi)\neq\kappa\beta(\Delta)};\kappa \vdash [\Phi \neq \Delta]\boxed{P} \quad \text{if } \neg(\beta(\Phi) =_\gamma \beta(\Delta))$$

$$[\text{sync}]\quad \beta;\gamma;\kappa \vdash \boxed{\overline{\Phi}\langle\Delta\rangle}.P \parallel \boxed{\Phi'(x)}.Q \xrightarrow{\tau} \beta[\beta(\Delta)/x];\gamma';\kappa' \vdash \overline{\Phi}\langle\Delta\rangle.\boxed{P} \parallel \Phi'(x).\boxed{Q}$$
$$\text{if } \beta(\Phi) \sim_\kappa \beta(\Phi'), \text{ with } \gamma' \stackrel{\text{def}}{=} \gamma_{\lhd\beta(\Phi)=\beta(\Phi')}, \kappa' \stackrel{\text{def}}{=} \kappa_{\lhd\beta(\Phi)=\beta(\Delta)}$$

$$[\text{sum}]\quad \beta;\gamma;\kappa \vdash \boxed{\textstyle\sum[P_1 + \ldots P_i \ldots + P_n]}.Q \xrightarrow{\epsilon} \beta;\gamma;\kappa \vdash \textstyle\sum[P_1 + \ldots \boxed{P_i} \ldots + P_n].Q$$

$$[\text{sum}_0]\quad \beta;\gamma;\kappa \vdash \textstyle\sum[P_1 + \ldots \boxed{\boxed{P_i}} \ldots + P_n].Q \xrightarrow{\varepsilon} \beta;\gamma;\kappa \vdash \textstyle\sum[P_1 + \ldots P_i \ldots + P_n].\boxed{Q}$$

$$[\text{par}]\quad \beta;\gamma;\kappa \vdash \boxed{\textstyle\prod[P_1 \parallel \ldots \parallel P_k]}.Q \xrightarrow{\varepsilon} \beta;\gamma;\kappa \vdash \textstyle\prod[\boxed{P_1} \parallel \ldots \parallel \boxed{P_k}].Q$$

$$[\text{par}_0]\quad \beta;\gamma;\kappa \vdash \textstyle\prod[\boxed{\boxed{P_1}} \parallel \ldots \parallel \boxed{\boxed{P_k}}].Q \xrightarrow{\varepsilon} \beta;\gamma;\kappa \vdash \textstyle\prod[P_1 \parallel \ldots \parallel P_k].\boxed{Q}$$

$$[\text{iter}]\quad \beta;\gamma;\kappa \vdash I : \boxed{*}P \xrightarrow{\varepsilon} \beta;\gamma;\kappa \vdash I : *\boxed{P}$$

$$[\text{iter}_0]\quad \beta;\gamma;\kappa \vdash I : *\boxed{\boxed{P}} \xrightarrow{\varepsilon} \mathsf{reset}^I(\beta);\gamma;\kappa \vdash I : \boxed{*}P$$

Now, the garbage collection function simply consists in removing all the occurrences of the inactive names in the context.

Definition 4. *In a pi-graph context $\beta;\gamma;\kappa$, let $E \stackrel{\text{def}}{=} \mathsf{inact}(\beta;\gamma;\kappa)$, then:*

$$\mathsf{gc}(\beta;\gamma;\kappa) \stackrel{\text{def}}{=} \beta;\gamma';\kappa' \text{ such that } \begin{cases} \gamma' \stackrel{\text{def}}{=} (\emptyset \lhd \{(x,y) \mid x \neq y \wedge x =_\gamma y \wedge x,y \notin E\}, \\ \qquad \{(x,y) \in \gamma^{\neq} \mid x,y \notin E\}) \\ \kappa' \stackrel{\text{def}}{=} \{(x, \kappa(x) \setminus E) \mid x \in \mathsf{dom}(\kappa) \setminus E\} \end{cases}$$

We now proceed to the individual description of the semantics rules.

The [silent] rule describes a subterm of π with a prefix τ as a redex. In such a situation, a transition through τ may be fired, leading to a state π' identical to π except that the continuation process P is activated.

The [par] rule is similar but the control-flow is now duplicated, simulating the fork of parallel processes. The latter works in conjunction with the [par_0] rule, which waits for all the forked processes to terminate before passing the control to the continuation Q. The non-deterministic choice among a set of possible

sub-processes is implemented by the [sum] and [sum$_0$] rules. Unlike parallel, only one branch is non-deterministically activated, and only the termination of this particular branch is required to activate the continuation of the choice.

The [iter] rule explains the start of a new iteration of an iterator named I. At the end of the iteration the rule [iter$_0$] reactivates the iterator so that a next iteration can start. The important step is the reinitialization of the names instantiated locally during the last iteration (cf. below).

As an illustration, we consider the example of an infinite generator of fresh names:

$$\{\nu^I a \mapsto \nu^I a\}; \emptyset; \emptyset \vdash I : \boxed{*}\ \overline{c}\langle \nu^I a \rangle.0$$

The only applicable rule is [iter] to start a new iteration, which yields:

$$\xrightarrow{\epsilon} \{\nu^I a \mapsto \nu^I a\}; \emptyset; \emptyset \vdash I : * \boxed{\overline{c}\langle \nu^I a \rangle}.0$$

We are now in a state where a private name must be sent over a public channel, a situation handled by the [o-fresh] rule. An important remark is that after the emission, the name $\nu^I a$ cannot be considered private anymore but *shared*. We implement the sharing of a private name by the instantiation of a fresh output name in set $\mathcal{N}_o$. To ensure freshness, we use the clock component κ of the context. The identity of the name is denoted $\mathsf{next_o}(\kappa)!$. In Table 1 we see that the principle is to take the least strictly positive integer that is not already in the domain of κ. Moreover, the clock is updated so that the newly generated identity is recorded, which is denoted $\mathsf{out}(\kappa)$ and simply consists in adding the new output name in the domain of the clock. In our running example, the clock is initially empty so the generated identity is 1!. The whole transition is depicted below:

$$\xrightarrow{\overline{c}\langle 1! \rangle} \{\nu^I a \mapsto 1!\}; \emptyset; \{1! \mapsto \emptyset\} \vdash I : * \overline{c}\langle \nu^I a \rangle.\boxed{0}$$

In this state the iteration terminates by [iter$_0$] and we must not forget to reinitialize the local private name $\nu^I a$ and apply garbage collection, which gives:

$$\xrightarrow{\epsilon} \{\nu^I a \mapsto \nu^I a\}; \emptyset; \emptyset \vdash I : \boxed{*}\ \overline{c}\langle \nu^I a \rangle.0$$

We are back in the initial state and we remark that the infinite fresh name generator is characterized finitely.

The rule [out] is a simpler variant triggered when a process emits a public name on a public channel. The rule [i-fresh] (fresh input) is quite similar to [o-fresh]. When a name is received from the environment, a fresh identity $\mathsf{next_i}(\kappa)?$ is generated for it. The clock is updated using $\mathsf{in}(\kappa)$ which adds the fresh input in the co-domains of all the existing output names (i.e. the domain of κ). This records *read-write causality* [10] that we illustrate together with the [match] rule below.

$$\beta, \nu^I a \mapsto \nu^I a, x^I \mapsto x^I; \emptyset; \emptyset \vdash I : * \boxed{\overline{c}\langle \nu^I a \rangle}.d(x^I).[\nu^I a = x^I]\ P$$
$$\xrightarrow{\overline{c}\langle 1! \rangle} \beta, \nu^I a \mapsto 1!, x^I \mapsto x^I; \emptyset; \{1! \mapsto \emptyset\} \vdash I : *\overline{c}\langle \nu^I a \rangle.\boxed{d(x^I)}.[\nu^I a = x^I]\ P$$
$$\xrightarrow{\overline{d}\langle 1? \rangle} \beta, \nu^I a \mapsto 1!, x^I \mapsto 1?; \emptyset; \{1! \mapsto \{1?\}\} \vdash I : *\overline{c}\langle \nu^I a \rangle.d(x^I).\boxed{[\nu^I a = x^I]}\ P$$

In the first step the [o-fresh] rule is triggered because we send a private name on a public channel. In consequence the private name $\nu^I a$ is instantiated by a fresh output 1!. The clock is also updated to record this fact. In the second step the [i-fresh] rule instantiates for x^I the fresh input 1?. The read-write causal link between the output 1! and the subsequent input 1? is recorded in the clock. The rationale is that it is indeed possible to receive an instance of 1! as 1? because the former is shared. Actually, the two names may be equated, which in term of Table 1 would be written $1!\overset{\gamma}{\sim}_\kappa 1?$ because $1! \prec_\kappa 1?$ (i.e. $1? \in \kappa(1!)$) where κ is the current clock and γ the partition. In the next step, the [match] rule is enabled and thus we can infer the following transition:

$$\xrightarrow{\epsilon} \beta'; \{1! = 1?\}; \{1! \mapsto \{1?\}\} \vdash I : *\overline{c}\langle \nu^I a \rangle.d(x^I).[\nu^I a = x^I] \boxed{P}$$
(where the instantiations β' remain unchanged)

The match has been effected and in the continuation P (left undetailed), the names 1! and 1? (and thus the occurrences of $\nu^I a$ and x^I) are considered equal. If we inverse the input and the output, then the input 1? will appear before the output 1!, and thus in the clock the two names will not be related. In consequence the final match would not be enabled and P could not be reached. We refer to [8] for a more thorough discussion about these non-trivial aspects.

The proposed dynamic interpretation of match suggests a similar treatment for mismatch, which is implemented by the [miss] rule. First, a mismatch between two names x, y is only possible if they are not provably equal (i.e. $x =_\gamma y$). There are then more possibilities for x, y to be inequal (denoted $x \neq^\kappa_\gamma y$). First they are inequal if either the name are explicitly distinguished in $\gamma^{\neq}$ or they are distinct private names. Another case is if one name is a (public) free name (in $\mathcal{N}_f$) and the other one is a private or an output name. The most non-trivial case is if one name is an input and the other one is an output, which makes them provably distinct if they are not causally related in κ (hence the causal clock is required). Finally, we must be careful not to forget about the interaction between equality and inequality. In all the other cases the mismatch leads to the explicit addition of a discriminating pair in $\gamma_{\neq}$.

Finally, the rule [sync] is for a communication taking place internally in a pi-graph. The subterm triggering the rule is composed of a pair of redexes: an output in one process and an input in another process (potentially from two distinct iterators). The effect of the rule is a communication from the output to the input process, the received name being instantiated in β. Because we need to characterize open systems, a non-trivial aspect here is that the communication can be triggered on distinct channel names, under the constraint that the match between the two names can be performed. This makes name matching and synchronization intimately related in the proposed semantics.

3.3 Abstracted Transitions

An important aspect of the pi-graphs semantics is the possibility to abstract away from low-level ϵ-transitions. In [8] we propose an inductive rule that allows

to omit the ϵ-transitions altogether, blindly. The main problem with this inductive principle is that it does not translate easily to the semantics of Petri nets. In this paper we use an alternative approach, which is to allow to branch directly on ϵ-transitions in non-deterministic choices. At first sight, this may lead to an incorrect situation where a deadlock may result from an unobservable ϵ-transition. To overcome this problem we introduce a *causal* principle of abstraction.

Definition 5. *Let* $\Gamma_1 \vdash \pi_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} \Gamma_n \vdash \pi_n \xrightarrow{\alpha_n} \Gamma_{n+1} \vdash \pi_{n+1}$ *be a sequence of transitions, with* θ_i, θ'_i *the subterms inducing the* i*-th transition (*$1 \leq i \leq n$*), following Definition 2. This sequence will be said* ***causal*** *iff* $\forall i,\ 1 \leq i < n,\ \exists j,\ i < j \leq n$ *such that* $\theta'_i \cap \theta_j \neq \emptyset$*, i.e., the* i*-th transition produces a redex (maybe many) needed by the* j*-th one.*

In such a causal sequence, the first transitions are all causally needed to produce the next ones, and in particular the last transition causally uses all the previous ones. The abstraction principle is then expressed as follows:

Definition 6. *An* ***abstracted transition*** $\Gamma_1 \vdash \pi_1 \overset{\alpha}{\dashrightarrow} \Gamma_n \vdash \pi_n$ *is inferred iff there is a causal sequence* $\Gamma_1 \vdash \pi_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} \Gamma_n \vdash \pi_n \xrightarrow{\alpha} \Gamma_{n+1} \vdash \pi_{n+1}$ $(\alpha \neq \epsilon)$.

The definition says that an abstracted transition with label α corresponds to a path of length n beginning with $n-1$ ϵ-transitions, and ending with a single non-ϵ-transition labelled α. Moreover, all the intermediate invisible transitions are causally needed to produce the visible one. This abstracts from invisible transitions and gets rid of the invisible transitions not causally needed to perform a visible one, which excludes to choose a branch in a choice leading to a deadlock through ϵ transitions. The definition is sound because we show in [8] that all *varepsilon*-paths have a bounded length.

4 Translation to Petri Nets

The key component of our proposition is the translation of the pi-graphs and their semantics into (concise) Petri nets. As a matter of fact, the pi-graph semantics has been designed to implicitly address the most complex issues of such a translation (control-flow, freshness, read-write causality, etc.). In this section we thus mostly assemble the pieces of the puzzle.

4.1 Petri Net Class and Transition Rule

Our translation requires a relatively simple class of coloured Petri nets. As usual in coloured models, places have types, arcs have annotations with variables and constants, and transitions have labels and guards. A particularity of these nets is that they accept only 1-safe markings. More precisely each place always has a unique token (possibly the "empty" token). Moreover, all the arcs are bidirectional and labelled by pairs R/W intended to *read* from the adjacent place a token $\rho(R)$ and *write* an updated token $\rho(W)$ (where ρ is a *binding* function for the variables in adjacent arcs and in the guard of the transition).

Definition 7. *A (coloured) Petri net is a tuple* $N = (S, T, U, G; M)$*, where*

- S *and* T *are the sets of places and transitions with* $S \cap T = \emptyset$*;*
- $S \times T$ *is the set of bidirectional arcs;*
- U *is a labeling mapping for each element of* $S \cup T \cup (S \times T)$*, such that*
  - *for each place* $s \in S$*,* $U(s)$ *gives its type (the set of admissible tokens);*
  - *for each transition* $t \in T$*,* $U(t)$ *is its (possibly empty) label;*
  - *for each arc in* $S \times T$*,* $U((s,t))$ *is a pair* R/W*, where* R *and* W *are constants or variables (or tuples thereon) compatible with* $U(s)$*.*
- G *is the mapping associating a guard (Boolean formula) to each* $t \in T$*;*
- M *is the marking associating to each place* $s \in S$ *a unique token in* $U(s)$*.*

As usual in high-level Petri nets, a transition $t \in T$ is *enabled* at a marking M iff there exists a *binding* ρ for the variables in its guard and in the arc inscriptions adjacent to t, such that ρ is compatible with the type of each place s adjacent to t, matches each token of $M(s)$ and makes the guard $G(t)$ true. The *occurrence* of t under ρ produces a new marking M' by consuming the tokens in all places adjacent to t and producing the new ones according to the arc annotations and conditions expressed in the guard. Such an occurrence of t is denoted $M[t : \rho > M'$.

4.2 Translation

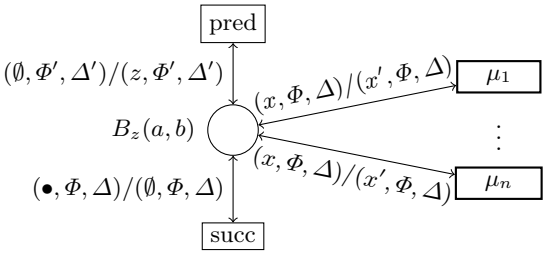
Let $\mathcal{E} \stackrel{\text{def}}{=} \beta_0; \gamma_0; \kappa_0 \vdash \pi_0$ be a pi-graph in its initial state (cf. Section 3.1). The translation is performed in two steps:

1. first, a *term net* $\mathfrak{T}(\pi_0)$ is obtained from a syntax-driven translation of π_0;
2. then, the translation $\mathsf{Pnet}(\mathcal{E})$ is obtained by composing $\mathfrak{T}(\pi_0)$ with a generic *context net* $\mathfrak{C}$, and by associating to the resulting net structure an initial marking.

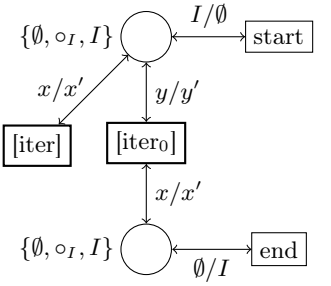
Step 1. The syntax-driven translation of pi-graph terms is based on a reduced set of basic *control-flow nets* that can be composed together using only three operators: (trivial) relabelling, disjoint union and merge. Table 3 gives the complete definition of the translation.

The *monadic control-flow nets*, denoted $\mathfrak{N}_z(a, b, \mu_1, \ldots, \mu_n)$, have only a single predecessor transition, labelled *pred*, and a single successor labelled *succ*. The control-flow is obtained from the predecessor and forwarded to the successor. The unique place of the net is connected to a set of *context transitions* $\mu_1, \ldots, \mu_n$ that correspond to rule names in the operational semantics of Table 2 (cf. the context net in Step 2 below). In the translation, the monadic nets are used to directly encode the constructs that reference names: input/output and match/mismatch. The set of context transition labels correspond to all the semantic rules that are potentially enabled when these constructs are in redex position, e.g. [i-fresh] and [sync] for the input prefix (cf. Table 2 and Figure 1). The place type of a monadic net is $B_z(a, b) \stackrel{\text{def}}{=} \{(z, a, b), (\bullet, a, b), (\emptyset, a, b)\}$, with a, b names in $\mathcal{N}$, and $z = \circ_i$ for an input prefix, $z = \circ_o$ for an output, and $z = \circ$ for match and mismatch.

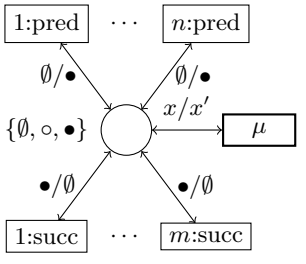
Table 3. Term nets – Step 1 of the translation



Monadic net $\mathfrak{N}_z(a, b, \mu_1, \ldots, \mu_n)$



Iterator net $\mathfrak{I}(I)$



Polyadic net $\mathfrak{M}(n, m, \mu)$
(if $n = 1$ or $m = 1$, then the corresponding indices in transition labels are removed)

$$
\begin{array}{l}
\mathfrak{T}([I_1 : *P_1 \parallel \ldots \parallel I_k : *P_k]) \stackrel{\text{def}}{=} \bigoplus_{j=1}^{k} \mathfrak{T}(I_j : *P_j) \\
\mathfrak{T}(I : *P) \stackrel{\text{def}}{=} \mathfrak{I}(I) \boxdot_{\{\text{start} \bowtie \text{pred}, \text{end} \bowtie \text{succ}\}} \mathfrak{T}(P) \\
\mathfrak{T}(p0) \stackrel{\text{def}}{=} \mathfrak{T}(p) \\
\mathfrak{T}(pP) \stackrel{\text{def}}{=} \mathfrak{T}(p) \boxdot_{\{\text{succ} \bowtie \text{pred}\}} \mathfrak{T}(P) \\
\mathfrak{T}(\tau) \stackrel{\text{def}}{=} \mathfrak{M}(1, 1, [\text{silent}]) \\
\mathfrak{T}(\overline{a}\langle b \rangle) \stackrel{\text{def}}{=} \mathfrak{N}_{\circ_o}(a, b, [\text{out}], [\text{o-fresh}], [\text{sync}]) \\
\mathfrak{T}(a(b)) \stackrel{\text{def}}{=} \mathfrak{N}_{\circ_i}(a, b, [\text{i-fresh}], [\text{sync}]) \\
\mathfrak{T}([a = b]P) \stackrel{\text{def}}{=} \mathfrak{N}_{\circ}(a, b, [\text{match}]) \boxdot_{\{\text{succ} \bowtie \text{pred}\}} \mathfrak{T}(P) \\
\mathfrak{T}([a \neq b]P) \stackrel{\text{def}}{=} \mathfrak{N}_{\circ}(a, b, [\text{miss}]) \boxdot_{\{\text{succ} \bowtie \text{pred}\}} \mathfrak{T}(P) \\
\mathfrak{T}(\sum[P_1 + \ldots + P_k]) \stackrel{\text{def}}{=} \mathfrak{M}(1, k, [\text{sum}]) \boxdot_A (\bigoplus_{j=1}^{k} j{:}\mathfrak{T}(P_j)) \boxdot_A \mathfrak{M}(k, 1, [\text{sum}_0]) \\
\qquad\qquad \text{where } A \stackrel{\text{def}}{=} \{j{:}\text{succ} \bowtie j{:}\text{pred} \mid j \in [1, k]\} \\
\mathfrak{T}(\prod[P_1 \parallel \ldots \parallel P_k]) \stackrel{\text{def}}{=} \mathfrak{M}(1, 1, [\text{par}]) \boxdot_B (\bigoplus_{j=1}^{k} j{:}\mathfrak{T}(P_j)) \boxdot_{B'} \mathfrak{M}(1, 1, [\text{par}_0]) \\
\text{where } B \stackrel{\text{def}}{=} \{\text{succ} \bowtie j{:}\text{pred} \mid j \in [1, k]\} \text{ and } B' \stackrel{\text{def}}{=} \{j{:}\text{succ} \bowtie \text{pred} \mid j \in [1, k]\}
\end{array}
$$

where the net operations are defined as follows:

- $j{:}N$ renames in N transition labels pred and succ to j:pred and j:succ;
- $\bigoplus_{j=1}^{k} N_j$ is a disjoint union of N_j's;
- $N \boxdot_A N'$, with $A \subseteq U(T) \times U'(T')$, is the disjoint union $N \oplus N'$ where for each pair $(a, a') \in A$, denoted $a \bowtie a'$, the transitions labeled a in N have been subsequently merged with the transitions labeled a' in N'; the merged transitions are anonymous; i.e., have empty labels and true guards.

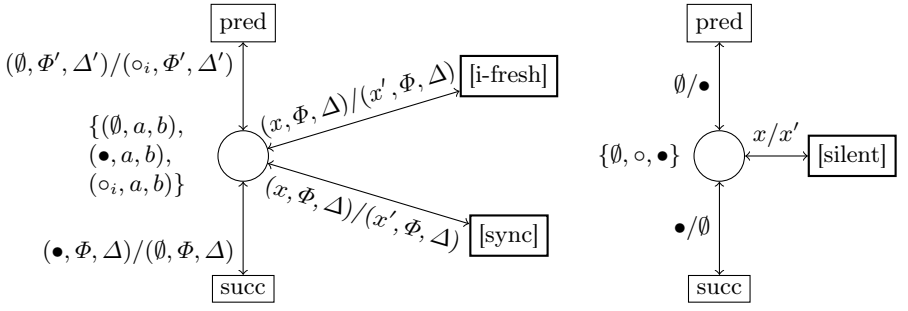


Fig. 1. Term net for Input $\mathfrak{T}(a(b))$ (left) and silent prefix $\mathfrak{T}(\tau)$ (right)

The control markers are: $\emptyset$ for inactive, $\circ_\lambda \in \{\circ, \circ_i, \circ_o\}$ for the activation of the place (redex position for the corresponding term), and $\bullet$ to trigger the successor transition (we illustrate the dynamics of the translated Petri net below).

The *polyadic control-flow nets*, denoted $\mathfrak{M}(n, m, \mu)$ are used to encode the remaining constructs but iterators. They have potentially multiple predecessors and successors but are connected to a single transition of the context net (e.g. [silent] for the τ prefix). The place type of polyadic nets is $\{\emptyset, \circ, \bullet\}$ with the same meaning as the monadic case for control markers. Finally, the *iterator nets* denoted $\mathfrak{I}(I)$ directly encode the iterator constructs. The type of the initial and final places of a translated iterator is $\{\emptyset, \circ_I, I\}$ where I is the iterator label and $\circ_I$ the control marker.

The translations for input and silent prefixes are illustrated in Figure 1. For the input prefix, the symbols Φ, Δ and Δ' are variables always bound to the translated channel name a and datum name b. The variables x and x' are used to modify the control-state of the place. This is managed by the context net (cf. Step 2 below). The other symbols are constants. The connection to the continuation of a prefix consists in merging the succ transition of the prefix with the pred transition of the next prefix (if it is a termination the prefix remains unmodified). The translation for output is similar to the input case. The match and mismatch are special prefixes (that cannot be suffixed by 0) but otherwise have quite similar translations. For parallel and sum, the entry and the exit places of the constructs have separate translations. A (polyadic) sum entry - connected to [sum] - has k successors (because a choice must be made) whereas the successor is unique for parallel entry (activation of all successor places at once through [par]). Symmetrically, the sum exit - connected to [sum_0] has k predecessors (only one must terminate) whereas it is unique for parallel exit (termination of all sub-processes at once through [par_0]). An iterator term net $\mathfrak{I}(I : * P)$ is obtained by the disjoint union of the translation of the iterator body $\mathfrak{T}(P)$ and the term net $\mathfrak{I}(I)$, with the explicit merge of the start/pred, and end/succ transitions. The translation of Example 2 is proposed in Figure 2.

Step 2. The term nets produced by the first step of the translation encode the basic control-structure of the pi-graphs. However, an extra layer is required to

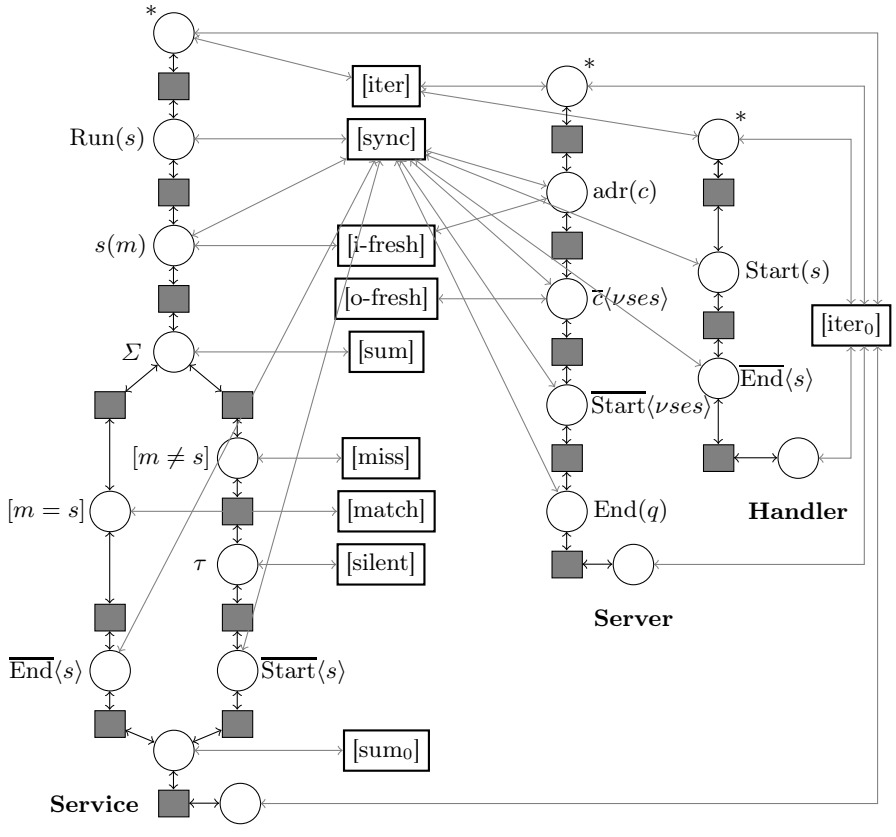
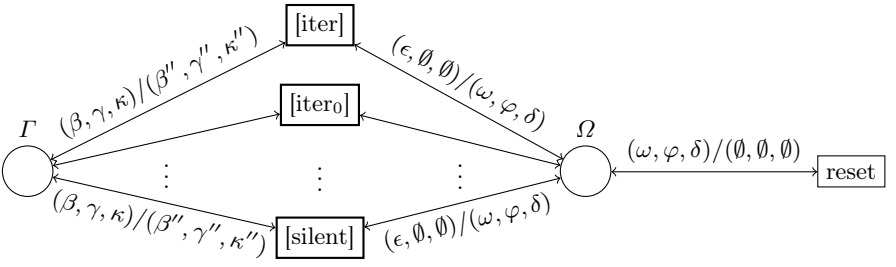


Fig. 2. Translation of the server specification (without place and arc labels)

orchestrate faithfully the semantics of Table 2. Technically speaking, we must connect the context transitions ([sync], [par], etc.) to a Petri net layer responsible of the orchestration: namely the *context net*. Given the relatively high-level and interleaving nature of the pi-graphs (as well as most pi-calculus variants), the role of the context net is to centralize the control so that each observation (i.e. a labelled transition in process calculus terms) can be generated *atomically*.

The generic context net $\mathfrak{C}$ - represented in Figure 3 - has only two places and fourteen transitions. The *global context place* Γ contains a single token (β, γ, κ) describing the current instantiations, partitions and causal clock (cf. Section 3). The *observation place* Ω contains informations about the current *observation* or $\emptyset$ for "unobservable" states. This place allows to abstract away from the internal states of the Petri nets and it also ensures the *mutual exclusion* among potential observations for the connected term net.

The reset transition is used to *discharge* the previous observation (by putting $(\emptyset, \emptyset, \emptyset)$ in Ω). The thirteen other transitions correspond to the thirteen rule names of Table 2. All the preconditions and side-effects of the semantic rules



$g([\text{silent}]) \stackrel{\text{def}}{=} \omega = \tau \wedge \exists u : x_u = \circ \wedge x'_u = \bullet$
$g([\text{out}]) \stackrel{\text{def}}{=} \omega = o \wedge \exists u(x_u = \circ_o \wedge \varphi = \beta(\Phi_u) \in \mathsf{Pub} \wedge \delta = \beta(\Delta_u) \in \mathsf{Pub} \wedge x'_u = \bullet)$
$g([\text{o-fresh}]) \stackrel{\text{def}}{=} \omega = o \wedge \delta = \mathsf{next_o}(\kappa)! \wedge \exists u(x_u = \circ_o \wedge \varphi = \beta(\Phi_u) \in \mathsf{Pub} \wedge$
$\beta(\Delta_u) \in \mathsf{Priv} \wedge x'_u = \bullet \wedge \beta' = \beta[\delta/\Delta_u]) \wedge \kappa' = \mathsf{out}(\kappa)$
$g([\text{i-fresh}]) \stackrel{\text{def}}{=} \omega = i \wedge \delta = \mathsf{next_i}(\kappa)? \wedge \exists u(x_u = \circ_i \wedge \varphi = \beta(\Phi_u) \in \mathsf{Pub} \wedge$
$x'_u = \bullet \wedge \beta' = \beta[\delta/\Delta_u]) \wedge \kappa' = \mathsf{in}(\kappa)$
$g([\text{match}]) \stackrel{\text{def}}{=} \omega = \epsilon \wedge \exists u(x_u = \circ \wedge \varphi = \beta(\Phi_u) \wedge \delta = \beta(\Delta_u) \wedge \varphi \tilde{\sim}_\kappa \delta \wedge x'_u = \bullet) \wedge$
$\gamma' = \gamma_{\lhd \varphi = \delta}$
$g([\text{miss}]) \stackrel{\text{def}}{=} \omega = \epsilon \wedge \exists u(x_u = \circ \wedge \varphi = \beta(\Phi_u) \wedge \delta = \beta(\Delta_u) \wedge \neg(\varphi =_\gamma \delta) \wedge x'_u = \bullet) \wedge$
$\gamma' = \gamma_{\lhd \varphi \neq^\kappa \delta}$
$g([\text{sync}]) \stackrel{\text{def}}{=} \omega = \tau \wedge \exists u, v(x_u = \circ_o \wedge x_v = \circ_i \wedge \beta(\Phi_u) \tilde{\sim}_\kappa \beta(\Phi_v) \wedge x'_u = x'_v = \bullet \wedge$
$\beta' = \beta[\beta(\Delta_u)/\Delta'_v] \wedge \gamma' = \gamma_{\lhd \beta(\Phi_u) = \beta(\Phi_v))}$
$g([\text{sum}]) = g([\text{sum}_0]) = g([\text{par}]) = g([\text{par}_0]) \stackrel{\text{def}}{=} \omega = \epsilon \wedge \exists u : x_u = \circ \wedge x'_u = \bullet$
$g([\text{iter}]) \stackrel{\text{def}}{=} \omega = \epsilon \wedge \exists u : x_u = \circ_I \wedge x'_u = I$
$g([\text{iter}_0]) \stackrel{\text{def}}{=} \omega = \epsilon \wedge \exists u : x_u \neq \emptyset \wedge \beta' = \mathsf{reset}^{x_u}(\beta) \wedge x'_u = \emptyset) \wedge \exists v : y_v = \emptyset \wedge y'_v = x_u)$
$g(\text{reset}) \stackrel{\text{def}}{=} (\omega, \varphi, \delta) \neq (\emptyset, \emptyset, \emptyset)$

Fig. 3. Generic context net $\mathfrak{C}$ and transition guards (if not explicitly indicated $x'_r = x_r$, $\beta'(r) = \beta(r)$, $\gamma' = \gamma$, $\kappa' = \kappa$, $\varphi = \emptyset$ and $\delta = \emptyset$) and $(\beta'', \gamma'', \kappa'') = \mathsf{gc}(\beta', \gamma', \kappa')$

are encoded as guards for the corresponding context transitions. The guards are listed in Figure 3 and the correspondence with the semantic rules of Table 2 is immediate. A context transition is enabled if the marking on Γ gives an appropriate global context, and if the marking on Ω is $(\emptyset, \emptyset, \emptyset)$ indicating that an observation is possible (mutual-exclusion). If an observation is enabled, the markings on Ω and Γ allow to retrieve it. For example, if the token on Ω is of the form (o, c, d), it means that the corresponding observation is the output $\overline{c}\langle d \rangle$, and analogously for the other actions. Firing the reset transition discharges the observation, allowing further observations.

The final net structure is a disjoint union of the nets $\mathfrak{T}(\pi)$ and $\mathfrak{C}$, in which:

- we remove from $\mathfrak{C}$ all the context transitions which are not present in $\mathfrak{T}(\pi)$;
- we merge all context transitions with the same label, and add to each resulting transition the corresponding guard.

Formally, this is defined as follows, using net operation $\mathfrak{T}(\pi) \Diamond_D \mathfrak{C}$, indexed by a set $D \stackrel{\text{def}}{=} \{(\mu, g(\mu)) \mid \mu \in \mathcal{M}\}$ of pairs (transition label, transition guard),

where $\mathcal{M}$ is the set of labels containing reset and all interface transition labels present in $\mathfrak{T}(\pi)$, and the guards $g(\mu)$ are as specified in Figure 3. For each pair $(\mu, g(\mu)) \in D$, let $t_1, \ldots, t_n$ be transitions with label μ in $\mathfrak{T}(\pi)$; for each arc (s, t_u), the variables in arc annotations $U(s, t_u)$ are indexed by u; all these transitions are merged together and with the transition with the same label in $\mathfrak{C}$; the resulting transition has label μ and guard $g(\mu)$.

The Petri net $\mathsf{Pnet}(\mathcal{E})$ is then obtained by associating to the structure resulting from Step 2 an initial marking M_0, which is $(\beta_0, \gamma_0, \kappa_0)$ on Γ, $(\emptyset, \emptyset, \emptyset)$ on Ω, $(\emptyset, a, b)$ on input $a(b)$, output $\overline{a}\langle b\rangle$, match $[a = b]$ and mismatch $[a \neq b]$ places, I_j on the initial place of iterator I_j, and $\emptyset$ on any other place.

4.3 Main Properties

The first important property of the proposed translation scheme is its *concision.*

Theorem 1. *The size of* $\mathsf{Pnet}(\mathcal{E})$ *is linear in the size of* $\mathcal{E}$

Proof. The proof is by inductive case analysis, with exactly one place and one transition for each prefix except sum and parallel (two places and one transition each), two places and one transition for iterators, and the context net has only two places and fourteen transitions. More precisely the size is bounded by $2i + p + 2s + 2$ places and $i + p + s + 14$ transitions (with i the number of iterators, p the number of simple prefixes and s the number of parallels and sums) □

The most important property we expect from the translation is that it agrees with the operational semantics of pi-graphs described in Section 3. Our objective is to show that each abstracted rewrite of a pi-graph is matched, in the translated Petri net, by an *abstracted occurrence* (defined below).

The Petri nets have lower-level semantics than the pi-graphs; in particular we may abstract away from any occurrence of an *anonymous transition* (i.e. a transition t such that $U(t) = \emptyset$). First, we may observe:

Lemma 1. *There is no infinite sequence of anonymous occurrences, and successive anonymous occurrences are causally independent.*

Proof. In the translation, we can easily see that between two anonymous transitions there is always a place connected to a context transition. As such, the context transition must be fired between the two anonymous occurrences. Of course, causally independent anonymous occurrences (in parallel processes or iterators) can still be performed in arbitrary order, but there can only be a finite number of these. □

Hence, we may consider sequences of occurrences $M_0[t_1 : \rho_1 > M_1 \ldots M_{n-1}[t_n : \rho_n > M_n$ such that $\forall i,\ 0 \leq i \leq n,\ U(t_i) = \emptyset$ and such that there is no anonymous transition t_{n+1} enabled from M_n. The markings $M_0, \ldots, M_n$ are considered as equivalent and the witness of the equivalence class is M_n, i.e. the (unique) state from which no further anonymous transition is enabled.

With this first level of abstraction, the Petri net markings are almost in one-to-one correspondence with pi-graph states.

Definition 8. *Let $\beta_1, \gamma_1, \kappa_1 \vdash \pi[\theta]$ be a pi-graph state with a subterm θ. We define $M^{\pi[\theta]}$ a marking of the corresponding Petri net such that $M^{\pi[\theta]}(\Gamma) = (\beta_1, \gamma_1, \kappa_1)$, $M^{\pi[\theta]}(\Omega) = (\emptyset, \emptyset, \emptyset)$ and each place corresponding to a redex in θ contains a control marker $\circ_\lambda$.*

The most important part of the agreement is that each pi-graph rewrite is matched by *exactly* two (abstract) Petri net occurrences: (1) the enabling of an observation, and (2) its discharge by the reset transition.

Lemma 2. *The rewrite $\beta_1; \gamma_1; \kappa_1 \vdash \pi[\theta] \xrightarrow{\alpha} \beta_2; \gamma_2; \kappa_2 \vdash \pi[\theta']$ can be inferred by rule μ of the semantics iff there exists a sequence of occurrences $M_1^{\pi[\theta]}[t_1 : \rho_1 > M'[t_2 : \rho_2 > M_2^{\pi[\theta']}$ such that $U(t_1) = \mu$, $U(t_2) = \mathsf{reset}$ and $\mathsf{label}(M'(\Omega)) = \alpha$*

with $\mathsf{label}(\omega, \varphi, \delta) \stackrel{\text{def}}{=} \varphi(\delta)$ *if* $\omega = i$, $\overline{\varphi}\langle\delta\rangle$ *if* $\omega = o$, ω *otherwise*

Proof. For the if part, we must take each rule of Table 2 and give its interpretation in terms of the translated Petri net. The translation has been designed to closely follow the rules, and thus all steps are almost direct and very similar. For the sake of concision, we only detail the case of the input prefix. The main hypothesis is a transition of the form:

$$\beta_1; \gamma_1; \kappa_1 \vdash \pi[\boxed{a(b)}.P] \xrightarrow{\beta_1(a)(\beta_1(b))} \mathsf{gc}(\beta_1[\mathsf{next}_\mathsf{i}(\kappa_1)?/\Delta]; \gamma_1; \mathsf{in}(\kappa_1)) \vdash \pi[a(b).\boxed{P}]$$

In Figure 4 we illustrate the two corresponding occurrences in the translated Petri net. The place named s_1 is the translation of the input prefix; initially this place is a redex since it contains the control marker $\circ_i$. The place s_2 is the first subterm of the continuation process P, its marking contains the control marker $\emptyset$, i.e. it is inactive. The global context Γ contains $(\beta_1, \gamma_1; \kappa_1)$ and there is no observation in Ω. In this situation the [i-fresh] transition can be fired, which leads to the second marking described in the figure. At the low-level, the [i-fresh] occurrence replaces the control marker $\circ_i$ by a continuation marker $\bullet$. However, this continuation is consumed by the anonymous transition between s_1 and s_2, which activates s_2. We only retain at the abstract level the state with s_1 marked $\emptyset$ (inactive) and s_2 marked with the control marker $\circ_\lambda$ (depending on the exact nature of s_2, it can be $\circ$, $\circ_i$ or $\circ_o$). The observation is captured by the marking of Ω. The context is also updated accordingly in the marking of Γ. Finally, the reset transition is fired to discharge the observation.

The only-if part consists in identifying each high-level occurrence of the Petri net to a corresponding pi-graph rule. There is no ambiguity since the Petri net provides the name of the rule (the label of the fired transition). Moreover, any observation must be discharged by the reset transition and thus the two steps are indeed atomic □

In the remainder we may forget about the particular subterm that is rewritten in the two-steps sequences identified by Lemma 2. In consequence, we simply denote a high-level occurrence as $M_1^\pi[\alpha > M_2^{\pi'}$. This allows a more general restatement of the previous Lemma as follows.

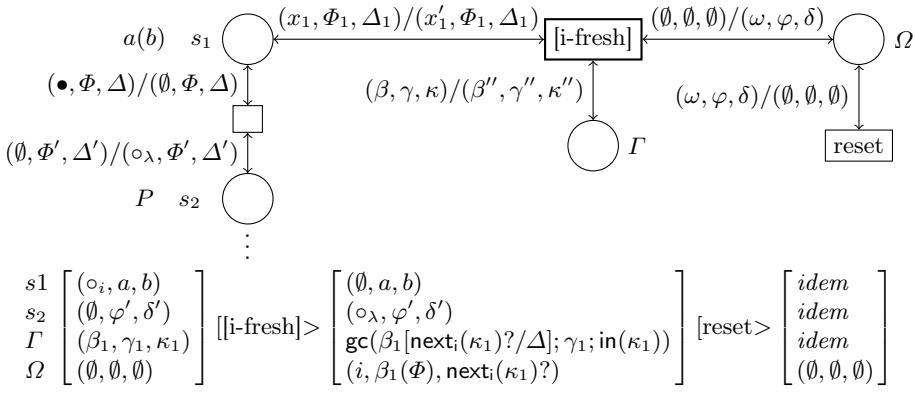


Fig. 4. Illustration of Lemma 2

Lemma 3. $\Gamma_1 \vdash \pi \xrightarrow{\alpha} \Gamma_2 \vdash \pi'$ *iff* $M_1^{\pi}[\alpha > M_2^{\pi'}$

The notion of abstracted transition (Definition 6) can then be reinterpreted.

Definition 9. *An* ***abstracted occurrence*** $M_1[\alpha \gg M_n$ *is produced iff there is a sequence* $M_1[\epsilon > M_2 \ldots M_{n-2}[\epsilon > M_n[\alpha > M_{n+1}$ $(\alpha \neq \epsilon)$ *with* $\forall i,\ 1 \leq i < n,\ \exists j,\ i < j \leq n$ *such that the* i*-th transition produces a redex token absorbed by the* j*-th one.*

This leads to the most important theorem of the study, as follows:

Theorem 2. ***(agreement)***
$\Gamma_1 \vdash \pi_1 \overset{\alpha}{\dashrightarrow} \Gamma_n \vdash \pi_n$ *iff* $M_1^{\pi_1}[\alpha \gg M_n^{\pi_n}$

Proof. For the only if part an abstracted transition (in the pi-graph) is assumed. By Lemma 3 we obtain directly a sequence of high-level occurrences (in the translated Petri net) verifying the conditions for an abstracted occurrence, as described above. Since Lemma 3 works both ways, the if part derives easily □

5 Related and Future Works

Compared to our previous work [5], we provide a simpler translation scheme to lower-level (but still coloured) Petri nets (without read arcs). The construct of iterator is preferred to general recursion in the pi-graphs. As demonstrated in [8], this provides a guarantee of finiteness by construction on the semantic side, a property only enjoyed for recursion-free processes in [5].

There are also various semantic translations of pi-calculi into Petri nets. Despite the different philosophy of these approaches based on reachability analysis, many interesting points of comparison remain. An example is [4], that investigates the translation of a variant of the pi-calculus into low-level P/T nets.

The supported language has general recursion but no match nor mismatch. The translation produces P/T nets whose places correspond to sub-processes (so-called *fragments*) in a normal structural form said restricted. The transitions are either *reactions* between reachable fragments or communications though public channels. The (interleaving) *reduction semantics* of the considered pi-calculus variant can be "fully retrieved" in the P/T nets, i.e. the formalisms *agree* in our own terms. Our translation reaches a similar agreement but in terms of *transition semantics*, a more complex setting. Indeed, without the match (and mismatch) and considering the reduction semantics only, our translation would be greatly simplified (e.g. no need for name partitions or causal clocks) and it would be interesting to see how far we get from P/T nets in this case.

Since the language of [4] has general recursion, it is possible to generate infinite systems. However, the semantic property of *structural stationarity* captures an interesting (albeit undecidable) subclass of potentially infinite systems that can be characterized finitely (by ensuring that there are only a finite number of reachable fragments up to structural congruence). In contrasts the pi-graphs cannot express processes with e.g. infinite control, which is a design choice because we require a guarantee of finiteness *by construction*.

Related semantic translations are proposed in [2,3], which interpret the (non-) interleaving and causal semantics for a pi-calculus in terms of P/T nets with inhibitor arcs. The translation provides an interpretation of the early transitions, whereas our interleaving semantic are closer to the late semantics. The advantage of the latter is that there is no need to study all the possible substitutions for the names generated by the environment. The authors of [3] identify the *finite net processes* that may only generate a bounded number of restricted names, a property that is not required in the pi-graphs thanks to the garbage collection of inactive names. A similar principle is proposed in the *history dependent automata* (HDA) framework [11]. The transitions of HDA provide injective correspondences between names, which ensures locally the freshness of the generated names. In the pi-graphs the freshness property is enforced at the global level by the use of the causal clock. One advantage of the global approach is the possibility to implement non-trivial phenomena such as read-write causality [10].

For the future works, we plan to take advantage of the *modular* structure of the translation with the term nets on the one side (interpretation of the syntax) and the context net on the other side (interpretation of the semantics). Variants of the context net (thus, of the semantics) could be considered to investigate concurrent and causal semantics. The Petri nets resulting from the translations are coloured but, we believe, with a lot of symmetry involved, which means efficient unfoldings could be produced for verification purpose.

References

1. Milner, R.: Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, Cambridge (1999)
2. Busi, N., Gorrieri, R.: A Petri net semantics for π-calculus. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 145–159. Springer, Heidelberg (1995)

3. Busi, N., Gorrieri, R.: Distributed semantics for the pi-calculus based on Petri nets with inhibitor arcs. J. Log. Algebr. Program. 78, 138–162 (2009)
4. Meyer, R., Gorrieri, R.: On the relationship between π-calculus and finite place/Transition petri nets. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 463–480. Springer, Heidelberg (2009)
5. Devillers, R., Klaudel, H., Koutny, M.: A compositional Petri net translation of general π-calculus terms. Formal Asp. Comput. 20, 429–450 (2008)
6. Berger, M.: An interview with Robin Milner, `http://www.informatics.sussex.ac.uk/users/mfb21/interviews/milner/` (2003)
7. Peschanski, F., Bialkiewicz, J.-A.: Modelling and verifying mobile systems using π-graphs. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 437–448. Springer, Heidelberg (2009)
8. Peschanski, F., Klaudel, H., Devillers, R.: A decidable characterization of a graphical pi-calculus with iterators. In: Infinity. EPTCS, vol. 39, pp. 47–61 (2010)
9. Busi, N., Gabbrielli, M., Tennenholtz, M.: Comparing recursion, replication, and iteration in process calculi. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 307–319. Springer, Heidelberg (2004)
10. Degano, P., Priami, C.: Causality for mobile processes. In: Fülöp, Z. (ed.) ICALP 1995. LNCS, vol. 944, pp. 660–671. Springer, Heidelberg (1995)
11. Montanari, U., Pistore, M.: History-dependent automata: An introduction. In: Bernardo, M., Bogliolo, A. (eds.) SFM-Moby 2005. LNCS, vol. 3465, pp. 1–28. Springer, Heidelberg (2005)

The Mutex Paradigm of Concurrency

Jetty Kleijn[1] and Maciej Koutny[2]

[1] LIACS, Leiden University
P.O.Box 9512, NL-2300 RA Leiden, The Netherlands
kleijn@liacs.nl
[2] School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@ncl.ac.uk

Abstract. Concurrency can be studied at different yet consistent levels of abstraction: from individual behavioural observations, to more abstract concurrent histories which can be represented by causality structures capturing intrinsic, invariant dependencies between executed actions, to system level devices such as Petri nets or process algebra expressions. Histories can then be understood as sets of closely related observations (here step sequences of executed actions). Depending on the nature of the observed relationships between executed actions involved in a single concurrent history, one may identify different *concurrency paradigms* underpinned by different kinds of causality structures (e.g., the true concurrency paradigm is underpinned by causal partial orders with each history comprising all step sequences consistent with some causal partial order). For some paradigms there exist closely matching system models such as elementary net systems (EN-systems) for the true concurrency paradigm, or elementary net systems with inhibitor arcs (ENI-systems) for a paradigm where simultaneity of executed actions does not imply their unorderedness.

In this paper, we develop a system model fitting the least restrictive concurrency paradigm and its associated causality structures. To this end, we introduce ENI-systems with *mutex* arcs (ENIM-systems). Each mutex arc relates two transitions which cannot be executed simultaneously, but can be executed in any order. To link ENIM-systems with causality structures we develop a notion of process following a generic approach (semantical framework) which includes a method to generate causality structures from the new class of processes.

Keywords: concurrency paradigms, elementary net systems, inhibitor arcs, mutex arcs, semantical framework, step sequences, process and causality semantics.

1 Introduction

Concurrency can be studied at different levels of abstraction, from the lowest level dealing with individual behavioural runs (observations), to the intermediate level of more abstract concurrent histories which can be represented by causality structures (or order structures) capturing intrinsic (invariant) dependencies between executed actions, to the highest system level dealing with devices such as Petri nets or process algebra expressions. Clearly, different descriptions of concurrent systems and their behaviours at these distinct levels of abstractions must be consistent and their mutual relationships well understood.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 228–247, 2011.

Abstract concurrent histories can be understood as sets of closely related observations. In this paper, each observation will be a *step sequence* (or stratified poset) of executed actions. For example, Figure 1(a) depicts an EN-system generating three step sequences involving the executions of transitions a, b and c, viz. $\sigma_1 = \{a, b\}\{c\}$, $\sigma_2 = \{a\}\{b\}\{c\}$ and $\sigma_3 = \{b\}\{a\}\{c\}$. They can be seen as belonging to a single abstract history $\Delta_1 = \{\sigma_1, \sigma_2, \sigma_3\}$ underpinned by a causal partial order in which a and b are unordered and they both precede c. From our point of view it is also important to note that Δ_1 adheres to the *true concurrency paradigm* captured by the following general statement:

> Given two executed actions (e.g., a and b in Δ_1), they can be observed as simultaneous (e.g., in σ_1) $\Longleftrightarrow$ they can be observed in both orders (e.g., a before b in σ_2, and b before a in σ_3). (TRUECON)

Concurrent histories adhering to such a paradigm are underpinned by *causal partial orders*, in the sense that each history comprises *all* step sequences consistent with some causal partial order on executed actions. Elementary net systems [18] (EN-systems) provide a fundamental and natural system level model for the true concurrency paradigm. A suitable link between an EN-system and histories like Δ_1 can be formalised using the notion of a process or occurrence net [1,18]. Full consistency between the three levels of abstraction can then be established within a generic approach (the *semantical framework* of [14]) aimed at fitting together systems (nets from a certain class of Petri nets), abstract histories and individual observations.

Depending on the exact nature of relationships holding for actions executed in a single concurrent history, similar to (TRUECON) recalled above, [9] identified eight general concurrency paradigms, π_1–π_8, with true concurrency being another name for π_8. Another paradigm is π_3 characterised by (TRUECON) with $\Longleftrightarrow$ replaced by $\Longleftarrow$. This paradigm has a natural system level counterpart provided by elementary net systems with inhibitor arcs (ENI-systems). Note that inhibitor arcs (as well as activator arcs used later in this paper) are well suited to model situations involving testing for a specific condition, rather than producing and consuming resources, and proved to be useful in areas such as communication protocols [2], performance analysis [4] and concurrent programming [5].

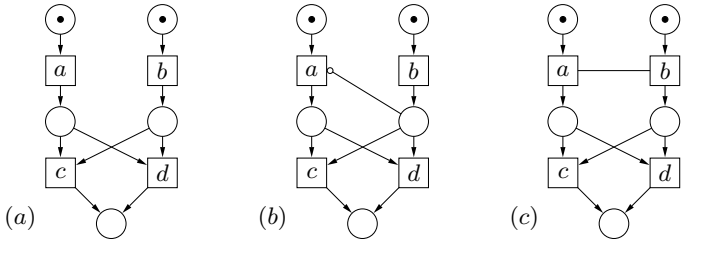


Fig. 1. EN-system (a); ENI-system with an inhibitor arc joining the output place of transition b with transition a implying that a cannot be fired if the output place of b is not empty (b); and ENIM-system with a mutex arc between transitions a and b implying that the two transitions cannot be fired in the same step (c)

For example, Figure 1(b) depicts an ENI-system generating two step sequences involving transitions a, b and c, viz. $\sigma_1 = \{a, b\}\{c\}$ and $\sigma_2 = \{a\}\{b\}\{c\}$. The two step sequences can be seen as belonging to the abstract history $\Delta_2 = \{\sigma_1, \sigma_2\}$ adhering to paradigm π_3, but *not* adhering to paradigm π_8 as there is no step sequence in Δ_2 in which b is observed before a (even though a and b are observed in σ_1 as simultaneous). Another consequence of the latter fact is that paradigm π_3 histories are underpinned *not* by causal partial orders but rather by causality structures introduced in [10] — called *stratified order structures* — based on causal partial orders and, in addition, weak causal partial orders. Again, full consistency between the three levels of abstraction can then be established within the semantical framework of [14].

In this paper, we focus on π_1 which simply admits all concurrent histories and is the least restrictive of the eight general paradigms of concurrency investigated in [9]. Concurrent histories conforming to paradigm π_1 are underpinned by yet another kind of causality structures introduced in [9] — called *generalised stratified order structures* — based on weak causal partial orders and *commutativity*. Intuitively, two executed actions commute if they may be observed in any order in step sequences belonging to a history, but they are never observed as simultaneous.

The aim of this paper is to develop the hitherto missing system level net model matching paradigm π_1. The proposed solution consists in extending ENI-systems with *mutex* arcs, where each mutex arc relates two transitions which cannot be executed simultaneously, even when they can be executed in any order. Mutex arcs are therefore a system level device implementing commutativity (for an early attempt aimed at capturing such a feature see [16]). The resulting ENIM-systems provide a natural match for histories conforming to paradigm π_1, in the same way as EN-systems and ENI-systems provided a natural match for histories conforming to paradigms π_8 and π_3, respectively.

For example, Figure 1(c) depicts an ENIM-system generating two step sequences involving transitions a, b and c, viz. $\sigma_2 = \{a\}\{b\}\{c\}$ and $\sigma_3 = \{b\}\{a\}\{c\}$. They belong to an abstract history $\Delta_3 = \{\sigma_2, \sigma_3\}$ adhering to paradigm π_1, in which the executions of a and b commute. Clearly, Δ_3 does *not* conform to paradigms π_8 and π_3 as there is no step sequence in Δ_3 in which a and b are observed as simultaneous.

We prove full consistency between the three levels of abstraction for paradigm π_1. To this end, we once more use the semantical framework of [14]. In doing so, we define processes of ENIM-systems and demonstrate that these new processes provide the desired link with the generalised stratified order structures of paradigm π_1. To achieve this we introduce a notion of gso-closure making it possible to construct generalised stratified order structures from more basic relationships between executed actions involved in processes of ENIM-systems. Note that ENIM-systems were first sketched in [12] however this preliminary presentation was still incomplete. In this paper, we provide the missing details and harmonise the treatment of paradigm π_1 with those of paradigms π_8 and π_3.

The paper is organised in the following way. To motivate our subsequent study of causality in nets with mutex arcs, we first briefly recall the approach of [9] which investigates general concurrency paradigms and the associated causality structures. We then recall the semantical framework of [14]. After that we formally introduce ENIM-systems and develop their process semantics. The paper concludes with the proofs of

various results which collectively justify our claim that ENIM-systems provide a fully satisfactory system model for paradigm π_1.

Basic Notions and Notations

Composing functions $f : X \to 2^Y$ and $g : Y \to 2^Z$ is defined by $g \circ f(x) \stackrel{\text{df}}{=} \bigcup_{y \in f(x)} g(y)$, for all $x \in X$. Restricting function f to a sub-domain Z is denoted by $f|_Z$. Relation $P \subseteq X \times X$ is irreflexive if $(x, x) \notin P$ for all $x \in X$; transitive if $P \circ P \subseteq P$; its transitive and reflexive closure is denoted by P^*; and its symmetric closure by $P^{\mathsf{sym}} \stackrel{\text{df}}{=} P \cup P^{-1}$.

A *relational structure* is a tuple $R \stackrel{\text{df}}{=} (X, Q_1, \ldots, Q_n)$ where X is a finite *domain*, and the Q_i's are binary relations on X (we can select its components using the subscript R, e.g., X_R). Relational tuples, R and R', are *isomorphic* if there is a bijection ξ from the domain of R to the domain of R' such that if we replace throughout R each element a by $\xi(a)$ then the result is R'. For relational structures with the same domain and arity, R and R', we write $R \subseteq R'$ if the subset inclusion holds component-wise. The intersection $\bigcap \mathcal{R}$ of a non-empty set $\mathcal{R}$ of relational structures with the same arity and domain is defined component-wise.

We *assume* that all sets in this paper are *labelled*, with the default labelling being the identity function. If the labelling is irrelevant for a definition or result, it may be omitted. If two domains are said to be the same, their labellings are identical.

A *partially ordered set* (or poset) is a relational structure $po \stackrel{\text{df}}{=} (X, \prec)$ consisting of a finite set X and a transitive irreflexive relation $\prec$ on X. Two distinct elements a, b of X are *unordered*, $a \frown b$, if neither $a \prec b$ nor $b \prec a$ holds. Moreover, $a \precsim b$ if $a \prec b$ or $a \frown b$. Poset po is *total* if the relation $\frown$ is empty, and *stratified* if $\simeq$ is an equivalence relation, where $a \simeq b$ if $a \frown b$ or $a = b$. Note that if a poset is interpreted as an observation of concurrent system behaviour, then $a \prec b$ means that a was observed before b, while $a \simeq b$ means that a and b were observed as simultaneous.

A *step sequence* is a sequence of non-empty sets $\sigma \stackrel{\text{df}}{=} X_1 \ldots X_k$ ($k \geq 0$). We will call σ *singular* if the steps X_i are mutually disjoint. In such a case, we have that $\mathsf{spo}(\sigma) \stackrel{\text{df}}{=} (\bigcup_i X_i, \bigcup_{i<j} X_i \times X_j)$ is a stratified poset. Conversely, each stratified poset spo induces a unique singular step sequence $\mathsf{steps}(spo) = X_1 \ldots X_k$, with each X_i being an equivalence class of $\simeq$ and $(X_i \times X_j) \subseteq \prec$ for all $i < j$, satisfying $spo = \mathsf{spo}(\mathsf{steps}(spo))$. We will identify each stratified poset spo with $\mathsf{steps}(spo)$ or, equivalently, each singular step sequence σ with $\mathsf{spo}(\sigma)$.

2 Paradigms of Concurrency and Order Structures

Let Δ be a non-empty set of *stratified posets* (or, equivalently, singular step sequences) with the same domain X (or X_Δ).[1] Intuitively, each poset in Δ is an observation of an abstract history of a hypothetical concurrent system. Following the true concurrency approach, [9] attempted to represent Δ using relational invariants on X. The basic idea was to capture situations where knowing some (or all) invariant relationships between

[1] Note that [9] also considered total and interval poset observations.

executed actions involved in Δ would be sufficient to reconstruct the entire set of observations Δ.

The approach of [9] identified a number of *fundamental invariants* which can be attributed to the observations in Δ, each invariant describing a relationship between pairs of executed actions which is repeated in all the observations of Δ. In particular, $\prec_\Delta$ comprises all pairs (a, b) such that a precedes b in every poset belonging to Δ; in other words, $\prec_\Delta$ represents *causality*. Other fundamental invariants are: $\rightleftharpoons_\Delta$ (*commutativity*, where $a \rightleftharpoons_\Delta b$ means that a and b are never simultaneous), $\sqsubset_\Delta$ (*weak causality*, where $a \sqsubset_\Delta b$ means that a is never observed after b) and $\bowtie_\Delta$ (*synchronisation*, where $a \bowtie_\Delta b$ means that a and b are always simultaneous). One can show that knowing $\rightleftharpoons_\Delta$ and $\sqsubset_\Delta$ is always sufficient to reconstruct Δ. This is done assuming that Δ is *invariant-closed* in the sense that Δ comprises all stratified posets *spo* with the domain X which respect all the fundamental invariants generated by Δ, e.g., $a \prec_\Delta b$ implies $a \prec_{spo} b$, and $a \sqsubset_\Delta b$ implies $a \precdot_{spo} b$. We then call each invariant-closed set of observations a *(concurrent) history*. Being invariant-closed is a natural assumption when constructing an abstract view of a possibly large set of individual observations, and has always been tacitly assumed in the causal partial order view of concurrent computation.

Depending on the underlying system model of concurrent computation, some additional constraints on histories Δ may be added. In particular, each design may adhere to the 'diagonal rule' — or 'diamond property' — by which simultaneity is the same as the possibility of occurring in any order, i.e., for all $a, b \in X$:

$$(\exists spo \in \Delta :\ a \frown_{spo} b) \iff (\exists spo \in \Delta :\ a \prec_{spo} b) \wedge (\exists spo \in \Delta :\ b \prec_{spo} a)\ . \quad (\pi_8)$$

For example, π_8 is satisfied by concurrent histories generated by EN-systems.

Constraints like π_8 — called *paradigms* in [8,9] — are essentially suppositions or statements about the intended treatment of simultaneity and, moreover, allow one to simplify the invariant representation of a history Δ. In particular, if Δ satisfies π_8 then one can reconstruct Δ using just causality $\prec_\Delta$ (which is always equal to the intersection of $\rightleftharpoons_\Delta$ and $\sqsubset_\Delta$). This is the essence of the true concurrency paradigm based on causal partial order.

In general, knowing $\prec_\Delta$ is insufficient to reconstruct Δ. For example, if we weaken π_8 to the paradigm:

$$(\exists spo \in \Delta :\ a \prec_{spo} b) \wedge (\exists spo \in \Delta :\ b \prec_{spo} a) \Longrightarrow (\exists spo \in \Delta :\ a \frown_{spo} b) \quad (\pi_3)$$

then one needs to enhance causality with weak causality $\sqsubset_\Delta$ to provide an invariant representation of Δ. The resulting relational structure $(X, \prec_\Delta, \sqsubset_\Delta)$ is an instance of the following notion.

Definition 1 (stratified order structure [6,11,13,14]). *A* stratified order structure *(or* SO-*structure) is a relational structure sos* $\stackrel{\text{df}}{=} (X, \prec, \sqsubset)$ *where* $\prec$ *and* $\sqsubset$ *are binary relations on* X *such that, for all* $a, b, c \in X$*:*

S1: $a \not\sqsubset a$ *S3:* $a \sqsubset b \sqsubset c \ \wedge\ a \neq c \implies a \sqsubset c$
S2: $a \prec b \implies a \sqsubset b$ *S4:* $a \sqsubset b \prec c \ \vee\ a \prec b \sqsubset c \implies a \prec c\ .$

The axioms imply that $\prec$ is a partial order relation, and that $a \prec b$ implies $b \not\sqsubset a$. The relation $\prec$ represents the 'earlier than' relationship on the domain of so, and the relation $\sqsubset$ the 'not later than' relationship. The four axioms capture the mutual relationship between the 'earlier than' and 'not later than' relations between executed actions.

For every stratified poset spo, $\mathsf{sos}(spo) \stackrel{\text{df}}{=} (X_{spo}, \prec_{spo}, \widehat{\prec}_{spo})$ is an SO-structure. Moreover, spo is a *stratified poset extension* of an SO-structure sos whenever $sos \subseteq \mathsf{sos}(spo)$. We denote this by $spo \in \mathsf{ext}(sos)$. Following Szpilrajn's Theorem [19] that any poset can be reconstructed by intersecting its total extensions, we have that any SO-structure can be reconstructed from its stratified poset extensions.

Theorem 1 ([11]). *If sos is an* SO*-structure then $\mathsf{ext}(sos) \neq \varnothing$ and:*

$$sos = \bigcap \{\mathsf{sos}(spo) \mid spo \in \mathsf{ext}(sos)\} \ .$$

Moreover, if $\mathcal{SPO}$ is a non-empty set of stratified posets with the same domain, then $\bigcap\{\mathsf{sos}(spo) \mid spo \in \mathcal{SPO}\}$ is an SO*-structure.* □

The set of stratified poset extensions of an SO-structure is a concurrent history satisfying paradigm π_3 [9]. Moreover, if a concurrent history Δ satisfies π_3, then $\Delta = \mathsf{ext}(X_\Delta, \prec_\Delta, \sqsubset_\Delta)$. Hence each abstract history Δ adhering to paradigm π_3 can be represented by the SO-structure $(X_\Delta, \prec_\Delta, \sqsubset_\Delta)$ [8].

If Δ fails to satisfy π_3, knowing $(X_\Delta, \prec_\Delta, \sqsubset_\Delta)$ may be insufficient to reconstruct Δ. In the case of paradigm π_1 which places no restrictions of the kind captured by π_8 or π_3 (i.e., Δ is only assumed to be invariant-closed), one needs to use *general* SO*-structures (*GSO*-structures)*.

Definition 2 (GSO-structure [7,8]). *A relational structure $gsos \stackrel{\text{df}}{=} (X, \rightleftharpoons, \sqsubset)$ is a* GSO-structure *if $\mathsf{sos}(gsos) \stackrel{\text{df}}{=} (X, \rightleftharpoons \cap \sqsubset, \sqsubset)$ is an* SO*-structure and the relation $\rightleftharpoons$ is symmetric and irreflexive.* ◇

In the above, $\rightleftharpoons$ represents the 'earlier than or later than, but never simultaneous' relationship, while $\sqsubset$ again represents the 'not later than' relationship.

For a stratified poset spo, $\mathsf{gsos}(spo) \stackrel{\text{df}}{=} (X_{spo}, \prec^{\mathsf{sym}}_{spo}, \widehat{\prec}_{spo})$ is a GSO-structure. Also, spo is a *stratified poset extension* of a GSO-structure $gsos$ if $gsos \subseteq \mathsf{gsos}(spo)$. We denote this by $spo \in \mathsf{ext}(gsos)$.

Each GSO-structure can be reconstructed from its stratified poset extensions, leading to another generalisation of Szpilrajn's Theorem.

Theorem 2 ([7,8]). *If $gsos$ is a* GSO*-structure then $\mathsf{ext}(gsos) \neq \varnothing$ and:*

$$gsos = \bigcap \{\mathsf{gsos}(spo) \mid spo \in \mathsf{ext}(gsos)\} \ .$$

Moreover, if $\mathcal{SPO}$ is a non-empty set of stratified posets with the same domain, then $\bigcap\{\mathsf{gsos}(spo) \mid spo \in \mathcal{SPO}\}$ is a GSO*-structure.* □

The set of stratified poset extensions of a GSO-structure is a concurrent history. Moreover, if Δ is a concurrent history, then $\Delta = \mathsf{ext}(X_\Delta, \rightleftharpoons_\Delta, \sqsubset_\Delta)$. Hence each abstract history Δ can be represented by the GSO-structure $(X_\Delta, \rightleftharpoons_\Delta, \sqsubset_\Delta)$ [8].

As already mentioned, paradigm π_8 and its associated causal posets $(X, \prec_\Delta)$ provide a match for concurrent histories generated by EN-systems. Similarly, one can show that paradigm π_3 and its associated SO-structures $(X, \prec_\Delta, \sqsubset_\Delta)$ provide a match for concurrent histories generated by ENI-systems. In this paper, we will extend ENI-systems with mutex arcs. The resulting ENIM-systems will provide a match for the most general paradigm π_1, and the notion of an abstract history of an ENIM-system will be captured through GSO-structures.

Constructing order Structures

We end this section describing ways of constructing SO-structures and GSO-structures from more basic, or direct, relationships. The idea is to proceed similarly as when constructing posets from acyclic relations through the operation of transitive closure. The definitions and results in this section are a new contribution to the theory of GSO-structures. Moreover, they are central for proving our subsequent results concerning nets with mutex arcs.

We first recall how the notion of transitive closure was lifted to the level of SO-structures. Let $\mu = (X, \prec, \sqsubset)$ be a relational structure (not necessarily an SO-structure). Intuitively, $\prec$ indicates which of the executed actions in X are directly causally related, and $\sqsubset$ which are directly weakly causally related. The *so-closure* of μ is defined as:

$$\mu^{\mathsf{so}} \stackrel{\text{df}}{=} (X, \alpha, \gamma \setminus id_X)$$

where $\gamma \stackrel{\text{df}}{=} (\prec \cup \sqsubset)^*$, $\alpha \stackrel{\text{df}}{=} \gamma \circ \prec \circ \gamma$ and id_X is the identity on X. Moreover, μ is *so-acyclic* if α is irreflexive. In such a case, μ^{so} is an SO-structure [10].

We will now show how to construct GSO-structures. Let $\rho = (X, \prec, \sqsubset, \rightleftharpoons)$ be a relational structure. In addition to the two relations appearing also in the μ above, $\rightleftharpoons$ indicates which of the executed actions may be observed in any order, but not simultaneously. The *gso-closure* of ρ is defined as:

$$\rho^{\mathsf{gso}} \stackrel{\text{df}}{=} (X, \psi, \gamma \setminus id_X)$$

where $\psi \stackrel{\text{df}}{=} \alpha^{\mathsf{sym}} \cup \beta^{\mathsf{sym}} \cup \rightleftharpoons$ with $\beta \stackrel{\text{df}}{=} \sqsubset^* \circ (\rightleftharpoons \cap \sqsubset^*) \circ \sqsubset^*$, in addition to α and γ being defined as for μ^{so}. Moreover, ρ is *gso-acyclic* if ψ is irreflexive and symmetric.

Proposition 1. *If ρ is gso-acyclic then ρ^{gso} is a* GSO*-structure.*

Proof. We first observe that (i) $\gamma = (\gamma \circ \prec \circ \gamma) \cup \sqsubset^*$, (ii) $\gamma = \gamma \circ \gamma$, and (iii) $\alpha \cup \beta \subseteq \gamma$. Moreover, (iv) $\alpha^{-1} \cap \gamma = \varnothing$ and (v) $\beta^{-1} \cap \gamma = \varnothing$. The two latter properties follow from (ii) and irreflexivity of α and β (which in turn follows from irreflexivity of ψ and $\alpha \cup \beta \subseteq \psi$).

Clearly, $\gamma \setminus id_X$ is irreflexive, and ψ is symmetric and irreflexive (by gso-acyclicity). Hence it suffices to show that $sos \stackrel{\text{df}}{=} (X, \psi \cap (\gamma \setminus id_X), \gamma \setminus id_X)$ is an SO-structure.

S1&S2: Clearly, $\gamma \setminus id_X$ is irreflexive, and $\psi \cap (\gamma \setminus id_X) \subseteq \gamma \setminus id_X$.

S3: $(\gamma \setminus id_X) \circ (\gamma \setminus id_X) \subseteq \gamma$ holds by (ii).

S4: We will show that $(\psi \cap (\gamma \setminus id_X)) \circ (\gamma \setminus id_X) \subseteq \psi \cap (\gamma \setminus id_X)$. Our first observation is that $\psi \cap (\gamma \setminus id_X) = \psi \cap \gamma$ as ψ is irreflexive. Hence it suffices to show that $(\psi \cap \gamma) \circ \gamma \subseteq \psi \cap \gamma$. We have that:

$$\psi \cap \gamma = (\alpha^{\mathsf{sym}} \cup \beta^{\mathsf{sym}} \cup \rightleftharpoons) \cap \gamma =_{(iv,v)} (\alpha \cup \beta \cup \rightleftharpoons) \cap \gamma =_{(iii)} \alpha \cup \beta \cup (\rightleftharpoons \cap \gamma)$$

which in turn implies that:

$$\begin{array}{l}
(\psi \cap \gamma) \circ \gamma = (\alpha \cup \beta \cup (\rightleftharpoons \cap \gamma)) \circ \gamma = \\
(\alpha \circ \gamma) \cup (\beta \circ \gamma) \cup ((\rightleftharpoons \cap \gamma) \circ \gamma) =_{(i,ii)} \\
\alpha \cup (\beta \circ ((\gamma \circ \prec \circ \gamma) \cup \sqsubset^*)) \cup ((\rightleftharpoons \cap \gamma) \circ ((\gamma \circ \prec \circ \gamma) \cup \sqsubset^*)) = \\
\alpha \cup (\beta \circ \gamma \circ \prec \circ \gamma) \cup (\beta \circ \sqsubset^*) \cup ((\rightleftharpoons \cap \gamma) \circ \gamma \circ \prec \circ \gamma) \cup ((\rightleftharpoons \cap \gamma) \circ \sqsubset^*) \subseteq \\
\alpha \cup \alpha \cup \beta \cup \alpha \cup ((\rightleftharpoons \cap \gamma) \circ \sqsubset^*) = \\
\alpha \cup \beta \cup ((\rightleftharpoons \cap ((\gamma \circ \prec \circ \gamma) \cup \sqsubset^*)) \circ \sqsubset^*) = \\
\alpha \cup \beta \cup ((\rightleftharpoons \cap (\gamma \circ \prec \circ \gamma)) \circ \sqsubset^*) \cup ((\rightleftharpoons \cap \sqsubset^*) \circ \sqsubset^*) \subseteq \\
\alpha \cup \beta \cup \alpha \cup \beta = \alpha \cup \beta \subseteq_{(iii)} \psi \cap \gamma \, .
\end{array}$$

As a result, *S4* holds as its other part is symmetric. □

Proposition 2. *If ρ is gso-acyclic then $(X, \prec, \sqsubset)$ is an so-acyclic relational structure and:*

$$\mathsf{ext}(\rho^{\mathsf{gso}}) = \{spo \in \mathsf{ext}((X, \prec, \sqsubset)^{\mathsf{so}}) \mid \frown_{spo} \cap \rightleftharpoons = \varnothing\} \, .$$

Proof. That $(X, \prec, \sqsubset)$ is so-acyclic follows immediately from irreflexivity of ψ and $\alpha \subseteq \psi$.

($\subseteq$) Let $spo \in \mathsf{ext}(\rho^{\mathsf{gso}})$. Then $\psi \subseteq \prec_{spo}^{\mathsf{sym}}$ and $\gamma \setminus id_X \subseteq \widehat{\prec}_{spo}$. Thus $\alpha^{\mathsf{sym}} \subseteq \prec_{spo}^{\mathsf{sym}}$ and $\gamma \setminus id_X \subseteq \widehat{\prec}_{spo}$ which together with $\alpha \subseteq \gamma$ and irreflexivity of α imply $\alpha \subseteq \prec_{spo}$. Hence $spo \in \mathsf{ext}((X, \prec, \sqsubset)^{\mathsf{so}})$. Moreover, we have $\rightleftharpoons \subseteq \prec_{spo}^{\mathsf{sym}}$ implying $\frown_{spo} \cap \rightleftharpoons = \varnothing$.

($\supseteq$) Let $spo \in \mathsf{ext}((X, \prec, \sqsubset)^{\mathsf{so}})$ and $\frown_{spo} \cap \rightleftharpoons = \varnothing$. Then $\alpha \subseteq \prec_{spo}$ and $\gamma \setminus id_X \subseteq \widehat{\prec}_{spo}$ and $\rightleftharpoons \subseteq \prec_{spo}^{\mathsf{sym}}$ (by $\frown_{spo} \cap \rightleftharpoons = \varnothing$ and irreflexivity of ψ and $\rightleftharpoons \subseteq \psi$). Therefore, it suffices to show that $\beta \subseteq \prec_{spo}$.

Suppose that $(a, b) \in \beta$. Then there are x, y such that $a \sqsubset^* x$ and $(x, y) \in \rightleftharpoons \cap \sqsubset^*$ and $y \sqsubset^* b$. By $(x, y) \in \rightleftharpoons$ and irreflexivity of ψ and $\rightleftharpoons \subseteq \psi$, we have that $x \neq y$. Thus, by the fact that $\sqsubset^* \setminus id_X \subseteq \gamma \setminus id_X \subseteq \widehat{\prec}_{spo}$ and $\frown_{spo} \cap \rightleftharpoons = \varnothing$, we have that $x \prec_{spo} y$. Moreover, again by $\sqsubset^* \setminus id_X \subseteq \gamma \setminus id_X \subseteq \widehat{\prec}_{spo}$, we have $a = x \vee a \, \widehat{\prec}_{spo} \, x$ and $y = b \vee y \, \widehat{\prec}_{spo} \, b$. Hence, since spo is a stratified poset, $a \prec_{spo} b$. □

The above result is similar to the following general characterisation of stratified poset extensions of GSO-structures.

Proposition 3. *If $gsos = (X, \rightleftharpoons, \sqsubset)$ is a* GSO*-structure then*

$$\mathsf{ext}(gsos) = \{spo \in \mathsf{ext}(\mathsf{sos}(gsos)) \mid \frown_{spo} \cap \rightleftharpoons = \varnothing\} \, .$$

Proof. ($\subseteq$) Let $spo \in \mathsf{ext}(gsos)$. Then $\rightleftharpoons \subseteq \prec_{spo}^{\mathsf{sym}}$ and $\sqsubset \subseteq \widehat{\prec}_{spo}$. Hence:

$$\rightleftharpoons \cap \sqsubset \; \subseteq \prec_{spo}^{\mathsf{sym}} \cap \widehat{\prec}_{spo} = \prec_{spo} \, ,$$

yielding $spo \in \mathsf{ext}(\mathsf{sos}(gsos))$. Moreover, $\frown_{spo} \cap \rightleftharpoons = \varnothing$, by $\rightleftharpoons \subseteq \prec_{spo}^{\mathsf{sym}}$.

$(\supseteq)$ Let $spo \in \mathsf{ext}(\mathsf{sos}(gsos))$ and $\frown_{spo} \cap \rightleftharpoons = \varnothing$. The latter and irreflexivity of $\rightleftharpoons$ implies $\rightleftharpoons \subseteq \prec_{spo}^{\mathsf{sym}}$. Moreover, $\sqsubset \subseteq \hat{\prec}_{spo}$, by $spo \in \mathsf{ext}(\mathsf{sos}(gsos))$. Hence $spo \in \mathsf{ext}(gsos)$. □

3 Fitting Nets and Order Structures

The operational and causality semantics of a class of Petri nets $\mathbb{PN}$ can be related within a common scheme introduced in [14]. It is reproduced here as Figure 2 where N is a net from $\mathbb{PN}$ and:

- $\mathbb{EX}$ are executions (or observations) of nets in $\mathbb{PN}$.
- $\mathbb{LAN}$ are labelled acyclic nets, each representing a history.
- $\mathbb{LEX}$ are labelled executions of nets in $\mathbb{LAN}$.
- $\mathbb{LCS}$ are labelled causal structures (order structures) capturing the abstract causal relationships between executed actions.

In this paper, the executions in $\mathbb{EX}$ step sequences, and the labelled executions in $\mathbb{LEX}$ are labelled singular step sequences.

The maps in Figure 2 relate the semantical views in $\mathbb{EX}$, $\mathbb{LAN}$, $\mathbb{LEX}$, and $\mathbb{LCS}$:

- ω returns a set of executions, defining the *operational* semantics of N.
- α returns a set of labelled acyclic nets, defining an *axiomatic process* semantics of N.
- π_N returns, for each execution of N, a non-empty set of labelled acyclic nets, defining an *operational process* semantics of N.
- λ returns a set of *labelled* executions for each process of N, and after applying ϕ to such labelled executions one should obtain executions of N.
- κ associates a labelled *causal* structure with each process of N.
- ϵ and $\imath$ allow one to go back and forth between labelled causal structures and sets of labelled executions associated with them.

The semantical framework captured by the above schema indicates how the different semantical views should agree. According to the rectangle on the left, the operational semantics of the Petri net defines processes satisfying certain axioms and moreover all labelled acyclic nets satisfying these axioms can be derived from the executions of the

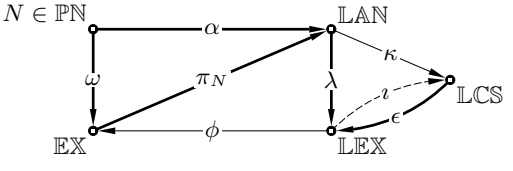


Fig. 2. Semantical framework for a class of Petri nets $\mathbb{PN}$. The bold arcs indicate mappings to powersets and the dashed arc indicates a partial function.

Petri net. Also, the labelled executions of the processes correspond with the executions of the original Petri net. The triangle on the right relates the labelled acyclic nets from $\mathbb{LAN}$ with the causal structures from $\mathbb{LCS}$ and the labelled executions from $\mathbb{LEX}$. The order structure defined by a labelled acyclic net can be obtained by combining executions of that net and, conversely, the stratified extensions of the order structure defined by a labelled acyclic net are its (labeled) executions. Thus the abstract relations between the actions in the labelled causal structures associated with the Petri net will be consistent with its chosen operational semantics.

To demonstrate that these different semantical views agree as captured through this semantical framework, it is sufficient to establish a series of results called *aims*. As there exist four simple requirements (called *properties*) guaranteeing these aims, one can concentrate on defining the semantical domains and maps appearing in Figure 2 and proving these properties.

Property 1 (soundness of mappings). *The maps* ω, α, λ, ϕ, $\pi_N|_{\omega(N)}$, κ, ϵ *and* $\imath|_{\lambda(\mathbb{LAN})}$ *are total. Moreover,* ω, α, λ, $\pi_N|_{\omega(N)}$ *and* ϵ *always return non-empty sets.* ◇

Property 2 (consistency). *For all* $\xi \in \mathbb{EX}$ *and* $LN \in \mathbb{LAN}$,

$$\left.\begin{array}{r} \xi \in \omega(N) \\ LN \in \pi_N(\xi) \end{array}\right\} \quad \text{iff} \quad \left\{\begin{array}{l} LN \in \alpha(N) \\ \xi \in \phi(\lambda(LN)) \end{array}\right. .$$

◇

Property 3 (representation). $\imath \circ \epsilon = id_{\mathbb{LCS}}$. ◇

Property 4 (fitting). $\lambda = \epsilon \circ \kappa$. ◇

The above four properties imply that the axiomatic (defined through α) and operational (defined through $\pi_N \circ \omega$) process semantics of nets in $\mathbb{PN}$ are in full agreement. Also, the operational semantics of N (defined through ω) coincides with the operational semantics of the processes of N (defined through $\phi \circ \lambda \circ \alpha$). Moreover, the causality in a process of N (defined through κ) coincides with the causality structure implied by its operational semantics (through $\imath \circ \lambda$). That is, we have the following.

Aim 1. $\alpha = \pi_N \circ \omega$. ◇

Aim 2. $\omega = \phi \circ \lambda \circ \alpha$. ◇

Aim 3. $\kappa = \imath \circ \lambda$. ◇

Thus, the operational semantics of the Petri net N and the set of labelled causal structures associated with it are related by $\omega = \phi \circ \epsilon \circ \kappa \circ \alpha$.

EN-Systems with Inhibitor Arcs

Usually, the fundamental net class for which processes and causality are introduced are EN-systems [18]. Here, however, we take elementary net systems with inhibitor arcs (ENI-systems) and use them to show how the semantical framework can be instantiated.

An ENI-system is a tuple $ENI \stackrel{\text{df}}{=} (P, T, F, Inh, M_{init})$ with P and T finite and disjoint sets of *places* — drawn as circles — and *transitions* — drawn as rectangles —, respectively; $F \subseteq (P \times T) \cup (T \times P)$ the flow relation of ENI — the directed arcs in the diagrams; $Inh \subseteq P \times T$ its set of *inhibitor* arcs — with small circles as arrowheads; and $M_{init} \subseteq P$ its initial marking. (In general, any subset of places is a *marking*, in diagrams indicated by small black dots.) If ENI has no inhibitor arcs, $Inh = \varnothing$, then it is an EN-system.

As usual, for every transition or place x we define its inputs ${}^\bullet x \stackrel{\text{df}}{=} \{y \mid (y, x) \in F\}$ and outputs $x^\bullet \stackrel{\text{df}}{=} \{y \mid (x, y) \in F\}$. Moreover, ${}^\circ t \stackrel{\text{df}}{=} \{p \mid (p, t) \in Inh\}$ are the inhibitor places of transition t. We also define for any subset U of T:

$$ {}^\bullet U \stackrel{\text{df}}{=} \bigcup_{t \in U} {}^\bullet t \ \text{ and } \ U^\bullet \stackrel{\text{df}}{=} \bigcup_{t \in U} t^\bullet \ \text{ and } \ {}^\circ U \stackrel{\text{df}}{=} \bigcup_{t \in U} {}^\circ t \ . $$

A *step of* ENI is a non-empty set U of transitions such that $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \varnothing$, for all distinct $t, u \in U$. A step U of ENI is *enabled* at a marking M of ENI if ${}^\bullet U \subseteq M$ and $(U^\bullet \cup {}^\circ U) \cap M = \varnothing$. Such a step can then be *executed* leading to the marking $M' \stackrel{\text{df}}{=} (M \setminus {}^\bullet U) \cup U^\bullet$. We denote this by $M[U\rangle_{ENI} M'$ or by $M[U\rangle M'$ if ENI is clear.

Thus the operational semantics of ENI is defined: $\omega(ENI)$ comprises all step sequences $\xi = U_1 \ldots U_k$ ($k \geq 0$) such that there are markings $M_{init} = M_0, \ldots, M_k$ with $M_{i-1}[U_i\rangle M_i$, for $i = 0, \ldots, k-1$. We call M_k a *reachable* marking of ENI.

In what follows we will assume that each inhibitor place p of an ENI-system ENI has a *complement place* $\widetilde{p}$ such that ${}^\bullet p = \widetilde{p}^\bullet$ and ${}^\bullet \widetilde{p} = p^\bullet$; moreover $|\{p, \widetilde{p}\} \cap M_{init}| = 1$. It is immediate that $|\{p, \widetilde{p}\} \cap M| = 1$, for all reachable markings M and all places p. Note that complement places can always be added to ENI as this does not affect its operational semantics.

Thus, for ENI-systems $\mathbb{EX}$ are step sequences. In addition, the labelled causal structures $\mathbb{LCS}$ are SO-structures, and the labelled executions $\mathbb{LEX}$ will be labelled singular step sequences. Next we introduce the labelled acyclic nets that will form the semantical domain $\mathbb{LAN}$ for the process semantics of ENI-systems. These nets will have activator rather than inhibitor arcs.

Definition 3 (activator occurrence nets). *An* activator occurrence net *(or* AO*-net) is a tuple* $AON \stackrel{\text{df}}{=} (P', T', F', Act, \ell)$ *such that:*

- *P', T' and F' are places, transitions and flow relation as in* ENI*-systems.*
- *$|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$, for every place p.*
- *$Act \subseteq P' \times T'$ is a set of activator arcs (indicated by black dot arrowheads) and ℓ is a labelling for $P' \cup T'$.*
- *The relational structure $\rho_{AON} \stackrel{\text{df}}{=} (T', \prec_{loc}, \sqsubset_{loc})$ is so-acyclic, where $\prec_{loc}$ and $\sqsubset_{loc}$ are respectively given by $(F' \circ F')|_{T' \times T'} \cup (F' \circ Act)$ and $Act^{-1} \circ F'$, as illustrated in Figure 3.* $\diamond$

We use ${}^\blacklozenge t \stackrel{\text{df}}{=} \{p \mid (p, t) \in Act\}$ to denote the activator places of a transition t, and ${}^\blacklozenge U \stackrel{\text{df}}{=} \bigcup_{t \in U} {}^\blacklozenge t$ for the activator places of a set $U \subseteq T'$. As for ENI-systems, a *step of* AON is a non-empty set U of transitions such that $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \varnothing$,

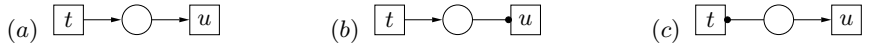


Fig. 3. Two cases (a) and (b) defining $t \prec_{loc} u$, and one case (c) defining $t \sqsubset_{loc} u$

for all distinct $t, u \in U$. A step U of AON is *enabled* at a marking M of AON if ${}^\bullet U \cup {}^\blacklozenge U \subseteq M$. The execution of such a U is defined as for ENI-systems and leads to the marking $(M \setminus {}^\bullet U) \cup U^\bullet$.

The default *initial* and *final* markings of AON are M_{init}^{AON} and M_{fin}^{AON} consisting respectively of all places p without inputs (${}^\bullet p = \varnothing$) and all places p without outputs ($p^\bullet = \varnothing$). The behaviour of AON is captured by the set $\lambda(AON)$ of all step sequences from M_{init}^{AON} to M_{fin}^{AON}. The set $\mathsf{reach}(AON)$ of markings *reachable* in AON comprises all markings M reachable from M_{init}^{AON} such that M_{fin}^{AON} is reachable from M. One can show that each step sequence $\sigma \in \lambda(AON)$ is singular, and that its set of elements is exactly the set of transitions T'. For such a step sequence σ, $\phi(\sigma)$ is obtained from σ by replacing each t by $\ell(t)$.

We define $\kappa(AON) \stackrel{\text{df}}{=} \rho_{AON}^{\mathsf{so}}$ which is guaranteed to be an SO-structure by the so-acyclicity of ρ_{AON} [10].

As far as the mappings ϵ and ι are concerned, ϵ is the set of stratified poset extensions (or, equivalently, singular step sequences) of an SO-structure, and ι is the intersection of the SO-structures (or, equivalently, singular step sequences) corresponding to a set of stratified posets with the same domain. Thus Theorem 1 immediately yields Property 3.

Finally, we give the axiomatic and operational process semantics of an ENI-system $ENI = (P, T, F, Inh, M_{init})$.

Definition 4 (processes of ENI-systems). *A* process *of ENI is an* AO*-net AON such that its labelling ℓ:*

- *labels the places of AON with places of ENI.*
- *labels the transitions of AON with transitions of ENI.*
- *is injective on M_{init}^{AON} and $\ell(M_{init}^{AON}) = M_{init}$.*
- *is injective on ${}^\bullet t$ and $t^\bullet$ and, moreover, $\ell({}^\bullet t) = {}^\bullet \ell(t)$ and $\ell(t^\bullet) = \ell(t)^\bullet$, for every transition t of AON.*
- *ℓ is injective on ${}^\blacklozenge t$ and $\ell({}^\blacklozenge e) = \widetilde{{}^\circ \ell(t)}$ for every transition t of AON.*

We denote this by $AON \in \alpha(ENI)$. $\diamond$

Definition 5 (processes construction). *An* AO*-net* generated *by a step sequence $\sigma = U_1 \ldots U_n \in \omega(ENI)$ is the last element in the sequence $AON_0, \ldots, AON_n$ where each $AON_k \stackrel{\text{df}}{=} (P_k, T_k, F_k, A_k, \ell_k)$ is an* AO*-net such that:*
Step 0: $P_0 \stackrel{\text{df}}{=} \{p^1 \mid p \in M_{init}\}$ and $T_0 = F_0 = A_0 \stackrel{\text{df}}{=} \varnothing$.
Step k: Given AON_{k-1} the sets of nodes and arcs are extended as follows:

$$
\begin{array}{rcl}
P_k & \stackrel{\text{df}}{=} & P_{k-1} \cup \{p^{1+\triangle p} \mid p \in U_k^\bullet\} \\
T_k & \stackrel{\text{df}}{=} & T_{k-1} \cup \{t^{1+\triangle t} \mid t \in U_k\} \\
F_k & \stackrel{\text{df}}{=} & F_{k-1} \cup \{(p^{\triangle p}, t^{1+\triangle t}) \mid t \in U_k \wedge p \in {}^\bullet t\} \\
 & & \quad \cup \{(t^{1+\triangle t}, p^{1+\triangle p}) \mid t \in U_k \wedge p \in t^\bullet\} \\
A_k & \stackrel{\text{df}}{=} & A_{k-1} \cup \{(\widetilde{p}^{\,\triangle \widetilde{p}}, t^{1+\triangle t}) \mid t \in U \wedge p \in {}^\circ t\}\ .
\end{array}
$$

In the above, the label of each node $\ell_k(x^i)$ is set to be x, and $\triangle x$ denotes the number of the nodes of AON_{k-1} labelled by x. We denote this by $AON_n \in \pi_{ENI}(\sigma)$. $\diamond$

Note that $\pi_{ENI}(\sigma)$ comprises exactly one net (up to isomorphism). The same holds for $\pi_{ENIM}(\sigma)$ defined later.

As one can show that the remaining properties are also satisfied, the semantical framework for ENI-systems holds [14].

4 Mutually Exclusive Transitions

We now introduce a new class of Petri nets by extending ENI-systems with mutex arcs prohibiting certain pairs of transitions from occurring simultaneously (i.e., in the same step). Consider Figure 4 which shows a variant of the producer/consumer scheme. In this case, the producer is allowed to retire (transition r), but never at the same time as the consumer finishes the job (transition f). Other than that, there are no restrictions on the executions of transitions r and f. To model such a scenario we use a mutex arc between transitions r and f (depicted as an undirected edge). Note that mutex arcs are relating transitions in a direct way. This should however not be regarded as an unusual feature as, for example, Petri nets with priorities also impose direct relations between transitions.

An *elementary net system with inhibitor and mutex arcs* (or ENIM-system) is a tuple $ENIM \stackrel{\text{df}}{=} (P, T, F, Inh, Mtx, M_{init})$ such that $\mathsf{und}(ENIM) \stackrel{\text{df}}{=} (P, T, F, Inh, M_{init})$ is the ENI-system *underlying* $ENIM$ and $Mtx \subseteq T \times T$ is a symmetric irreflexive relation specifying the *mutex* arcs of $ENIM$. Where possible, we retain the definitions introduced for ENI-systems. The notion of a step now changes however. A *step of* $ENIM$ is a non-empty set U of transitions such that U is a step of $\mathsf{und}(ENIM)$ and in addition $Mtx \cap (U \times U) = \varnothing$. With this modified notion of a step, the remaining definitions pertaining to the dynamic aspects of an ENIM-system, including $\omega(ENIM)$, are the same as for the underlying ENI-system $\mathsf{und}(ENIM)$.

Proposition 4. $\omega(ENIM) = \{U_1 \ldots U_k \in \omega(\mathsf{und}(ENIM)) \mid Mtx \cap \bigcup_i U_i \times U_i = \varnothing\}$.

Proof. Follows from the definitions. □

For the ENIM-system of Figure 4, we have that $M[\{r\}\rangle M''[\{f\}\rangle M'$ as well as $M[\{f\}\rangle M'''[\{r\}\rangle M'$, where $M = \{p_2, p_4, p_6\}$ and $M' = \{p_0, p_4, p_7\}$. However, $M[\{r, f\}\}\rangle M'$

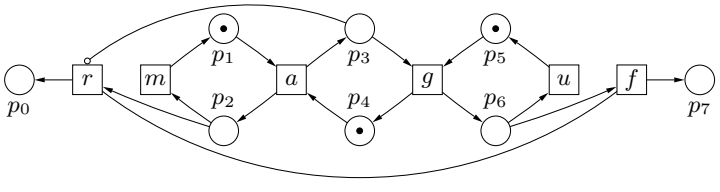


Fig. 4. An ENIM-system modelling a producer/consumer system with the actions: 'make item' m, 'add item to buffer' a, 'get item from buffer' g, 'use item' u, 'producer retires' r, and 'consumer finishes' f. Note: the producer can only retire if the buffer is empty (i.e., p_3 is empty).

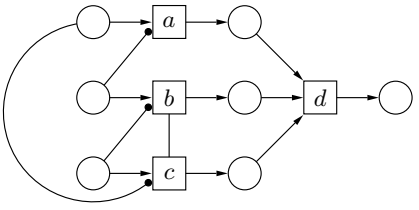


Fig. 5. A net which is not an AMO-net as it fails the gso-acyclicity test

which holds for the underlying ENI-system does not hold now as r and f cannot be executed in the same step.

To deal with the behaviours of ENIM-systems in the context of the semantical framework, we adapt the approach followed for ENI-system as recalled above. The labelled causal structures, $\mathbb{LCS}$, are now GSO-structures, while labelled executions, $\mathbb{LEX}$, are labelled singular step sequences, as before. The labelled acyclic nets, $\mathbb{LAN}$, used for the process semantics of ENIM-systems are introduced next.

Definition 6 (activator mutex occurrence nets). *An* activator mutex occurrence net *(or* AMO*-net) is a tuple* $AMON \stackrel{\text{df}}{=} (P', T', F', Act, Mtx', \ell)$ *such that:*

- $\mathsf{und}(AMON) \stackrel{\text{df}}{=} (P', T', F', Act, \ell)$ *is the* AO*-net* underlying $AMON$ *and* $Mtx' \subseteq T' \times T'$ *is a symmetric irreflexive relation specifying the* mutex *arcs of* $AMON$.
- $\rho_{AMON} \stackrel{\text{df}}{=} (T', \prec_{loc}, \sqsubset_{loc}, Mtx')$, *where* $\prec_{loc}$ *and* $\sqsubset_{loc}$ *are defined as for* AO*-nets in Definition 3, is a gso-acyclic relational structure.* ◇

The part of gso-acyclicity ρ_{AMON} which deals with the mutex arcs is illustrated in Figure 5. We have there three transitions satisfying $a \sqsubset_{loc} b \sqsubset_{loc} c \sqsubset_{loc} a$. Hence, in any execution involving all these transitions, they have to belong to the same step. This, however, is inconsistent with a mutex arc between b and c, and the gso-acyclicity fails to hold because (a, a) belongs to $\sqsubset_{loc}^* \circ (Mtx' \cap \sqsubset_{loc}^*) \circ \sqsubset_{loc}^*$.

Then we let $\kappa(AMON) \stackrel{\text{df}}{=} \rho_{AMON}^{\mathsf{gso}}$ be the GSO-structure generated by $AMON$. Note that Proposition 1 guarantees the correctness of this definition. Moreover, it is consistent with the SO-structure defined by its underlying AO-net.

Proposition 5. $(T', \prec_{loc}, \sqsubset_{loc})$ *is an so-acyclic relational structure.*

Proof. Follows from Proposition 2. □

As far as the mappings ϵ and ι are concerned, ϵ is the set of stratified poset (or, equivalently, singular step sequences) extensions of a GSO-structure, and ι is the intersection of the GSO-structures corresponding to a set of stratified posets with the same domain. Thus Theorem 2 immediately yields Property 3. Other properties are dealt with later in this section.

The default initial and final markings of $AMON$, as well as its step sequence executions are defined exactly the same as for the underlying AO-net under the proviso that steps do not contain transitions joined by mutex arcs.

The following results yield more insight into the labelled executions of an activator mutex occurrence net relative to its underlying AO-net.
Let $AMON = (P', T', F', Act, Mtx', \ell)$ be an AMO-net and $AON = \mathsf{und}(AMON)$.

Proposition 6. $\lambda(AMON) = \{U_1 \ldots U_k \in \lambda(AON) \mid Mtx' \cap \bigcup_i U_i \times U_i = \varnothing\}$.

Proof. Follows from the definitions. □

Proposition 7. *Let $\sigma = U_1 \ldots U_k \in \lambda(AON)$ be such that there is no $i \leq k$ for which there exists a partition U, U' of U_i such that $U_1 \ldots U_{i-1} U U' U_{i+1} \ldots U_k \in \lambda(AON)$. Then $\sigma \in \lambda(AMON)$.*

Proof. By Proposition 6, it suffices to show that, for every $i \leq k$, $(U_i \times U_i) \cap Mtx' = \varnothing$. Suppose this does not hold for some $i \leq k$. Let $\kappa(AON) = (T', \prec, \sqsubset)$. From the assumption made about σ it follows that $t \sqsubset u$, for all distinct $t, u \in U_i$. This, however, contradicts the gso-acyclicity of ρ_{AMON}. □

Proposition 8. $\mathsf{reach}(AMON) = \mathsf{reach}(AON)$.

Proof. ($\subseteq$) Follows from Proposition 6.
($\supseteq$) Follows from Proposition 7 and the fact that each step sequence in $\lambda(AON)$ can be 'sequentialised' into the form from the formulation of Proposition 7 by splitting the steps into smaller ones. □

Proposition 9. *A marking M belongs to* $\mathsf{reach}(AMON)$ *iff there are no places $p, p' \in M$ for which $(p, p') \in F' \circ (\prec_{loc} \cup \sqsubset_{loc})^* \circ F'$.*

Proof. Follows from Proposition 8 and Proposition 5.15 in [14]. □

Figure 6 depicts an AMO-net labelled with places and transitions of the ENIM-system of Figure 4. We have that both $\{a\}\{g\}\{r\}\{f\}$ and $\{a\}\{g\}\{f\}\{r\}$ belong to $\phi(\lambda(AMON_0))$, however, $\{a\}\{g\}\{f, r\}$ does not.

Now we are ready to introduce process semantics for ENIM-systems.

Definition 7 (processes of ENIM-systems). *A* process *of ENIM is an* AMO*-net $AMON$ such that* $\mathsf{und}(AMON)$ *is a process of* $\mathsf{und}(ENIM)$ *and, for all $t, u \in T'$, we have $(t, u) \in Mtx'$* iff *$(\ell(t), \ell(u)) \in Mtx$. We denote this by $AMON \in \alpha(ENIM)$.* ◇

Definition 8 (processes construction). *An* AMO*-net* generated *by a step sequence $\sigma = U_1 \ldots U_n \in \omega(ENIM)$ is the last net in the sequence $AMON_0, \ldots, AMON_n$ where each $AMON_k \stackrel{\text{df}}{=} (P_k, T_k, F_k, A_k, M_k, \ell_k)$ is as in Definition 5 except that $M_k \stackrel{\text{df}}{=} \{(e, f) \in T_k \times T_k \mid (\ell_k(e), \ell_k(f)) \in Mtx\}$ is an added component. We denote this by $AMON_n \in \pi_{ENIM}(\sigma)$* ◇

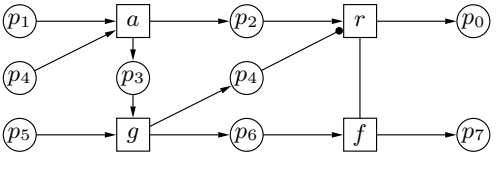


Fig. 6. An AMO-net $AMON_0$ with labels shown inside places and transitions

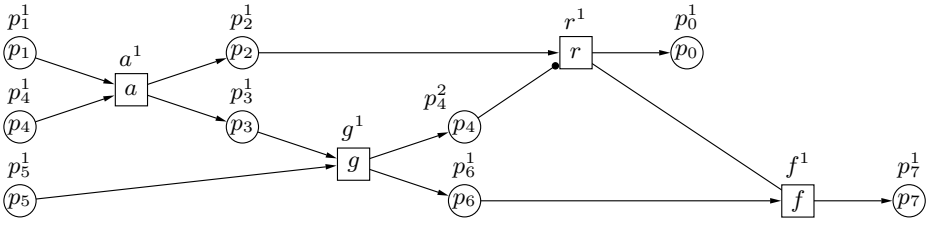


Fig. 7. Process generated for the ENIM-system in Figure 4 and $\sigma \stackrel{\text{df}}{=} \{a\}\{g\}\{r\}\{f\}$

The way in which mutex arcs are added in the process construction entails that some of them may be redundant when, for example, the transitions they join are causally related. However, eliminating such redundant mutex arcs (which is possible by analysing paths in the AMO-net) would go against the locality principle which is the basis of the process approach. Indeed, this approach does not remove redundant causalities as this would compromise the local causes and effects in the definition and construction of process nets.

The AMON-net shown in Figure 6 is a process of the ENIM-system of Figure 4 with $\phi(\lambda(AMON_0)) = \big\{\{a\}\{g\}\{f\}\{r\}, \{a\}\{g\}\{r\}\{f\}\big\}$. Figure 7 shows the result of applying the construction from Definition 8 to the ENIM-system of Figure 4 and one of its step sequences. Note that the resulting AMO-net is isomorphic to that shown in Figure 6.

Having instantiated the semantical framework for ENIM-systems, we can now formally establish their connection with GSO-structures by proving the remaining Properties 1, 2, and 4. Below we assume that $ENIM$ is an ENIM-system.

Proposition 10. *Let σ a step sequence of $ENIM$, $AMON$ an* AMO*-net, $gsos$ a* GSO*-structure, and $\mathcal{SPO}$ a set of stratified posets with the same domain.*

1. *$\omega(ENIM)$, $\alpha(ENIM)$, $\lambda(AMON)$ and $\epsilon(gsos)$ are non-empty sets.*
2. *$\kappa(AON)$ and $\iota(\mathcal{SPO})$ are* GSO*-structures.*
3. *$\pi_{ENIM}(\sigma)$ comprises an* AMO*-net.*

Proof. In what follows, we use the notations introduced throughout this section.

(1) We have $\omega(ENIM) \neq \varnothing$ as the empty string is a valid step sequence of $ENIM$. To show $\alpha(ENIM) \neq \varnothing$ one can take the AMO-net consisting of the initial marking of $ENIM$ with the identity labelling and no transitions. That $\epsilon(gsos) \neq \varnothing$ follows from Theorem 2. That $\lambda(AMON) \neq \varnothing$ follows from Proposition 7, $\lambda(AON) \neq \varnothing$ and the fact that each step sequence in $\lambda(AON)$ can be 'sequentialised' into the form from the formulation of Proposition 7 by splitting the steps into smaller ones.

(2) Follows from Theorem 2 and Proposition 1.

(3) We have that an element of $\pi_{ENIM}(\sigma)$ with deleted mutex arcs is an AO-net. It therefore suffices to show that the relation $\sqsubset_{loc}^* \circ (Mtx' \cap \sqsubset_{loc}^*) \circ \sqsubset_{loc}^*$ is irreflexive.

Suppose that $(t,t) \in \sqsubset_{loc}^* \circ (Mtx' \cap \sqsubset_{loc}^*) \circ \sqsubset_{loc}^*$. Then there are $t = t_1, \ldots, t_k = t$ such that $(t_i, t_{i+1}) \in \sqsubset_{loc}$ for all $i < k$, and $(t_m, t_j) \in M_n$ for some $m < j \leq k$.

But this means that $t_1, \ldots, t_k$ have been generated in the same step of the construction, contradicting the definition of executability in ENIM-systems. □

Proposition 11. *Let $\xi \in \omega(ENIM)$ and $AMON \in \pi_{ENIM}(\xi)$.*

1. $AMON \in \alpha(ENIM)$.
2. $\xi \in \phi(\lambda(AMON))$.

Proof. (1) By Proposition 10(3), $AMON$ is an AMO-net. Moreover, by [14], we have that $\mathsf{und}(AMON) \in \alpha(\mathsf{und}(ENIM))$. Finally, the condition involving mutex arcs follows from the construction in Definition 8.

(2) By [14], $\xi \in \phi(\lambda(\mathsf{und}(AMON)))$. Hence $\xi = \phi(\sigma)$ for some $\sigma = U_1 \ldots U_k \in \lambda(\mathsf{und}(AMON))$. The latter, together with $\xi \in \omega(ENIM)$ and the consistency between mutex arcs in $ENIM$ and $AMON$, means that there is no mutex arc joining two elements of any U_i. Hence, by Proposition 6, $\sigma \in \lambda(AMON)$. Thus $\xi \in \phi(\lambda(AMON))$. □

Proposition 12. *Let $AMON \in \alpha(ENIM)$ and $\xi \in \phi(\lambda(AMON))$.*

1. $\xi \in \omega(ENIM)$.
2. $AMON \in \pi_{ENIM}(\xi)$.

Proof. (1) By [14], $\xi \in \omega(\mathsf{und}(ENIM))$. Also there is $\sigma = U_1 \ldots U_k \in \lambda(AMON)$ such that $\xi = \phi(\sigma)$. The latter, together with the consistency between mutex arcs in $ENIM$ and $AMON$, means that there is no mutex arc joining two elements of any U_i. Hence, by Proposition 4, $\xi \in \omega(ENIM)$.

(2) By [14], $\mathsf{und}(AMON) \in \pi_{\mathsf{und}(ENIM)}(\xi)$. Moreover, the mutex arcs are added in the same (deterministic) way to the underlying process nets, leading to $AMON \in \pi_{ENIM}(\xi)$. □

Hence Property 2 holds. We then observe that Property 3 is simply Theorem 2, and Property 4 is proved below.

Proposition 13. *Let $AMON$ be an AMO-net. Then $\lambda(AMON) = \epsilon(\kappa(AMON))$.*

Proof. We have:

$$\begin{array}{l}
\epsilon(\kappa(AMON)) = \mathsf{ext}(\rho^{\mathsf{gso}}_{AMON}) = \mathsf{ext}((T', \prec_{loc}, \sqsubset_{loc}, Mtx')^{\mathsf{gso}}) =_{\text{(Prop. 2)}} \\
\{spo \in \mathsf{ext}((T', \prec_{loc}, \sqsubset_{loc})^{\mathsf{so}}) \mid \frown_{spo} \cap Mtx' = \varnothing\} = \\
\{spo \in \epsilon(\kappa(AON)) \mid \frown_{spo} \cap Mtx' = \varnothing\} = \\
\{spo \in \lambda(AON) \mid \frown_{spo} \cap Mtx' = \varnothing\} =_{\text{(Prop. 6)}} \lambda(AMON).
\end{array}$$

Note that we identify stratified posets with their corresponding singular labelled step sequences. □

Finally, we can claim the semantical aims for ENIM-systems.

Theorem 3. *Let $ENIM$ be an ENIM-system, and $AMON$ be an AMO-net.*

$$\begin{array}{rcl}
\alpha(ENIM) & = & \pi_{ENIM}(\omega(ENIM)) \\
\omega(ENI) & = & \phi(\lambda(\alpha(ENIM))) \\
\kappa(AMON) & = & \iota(\lambda(AMON)) \; .
\end{array}$$

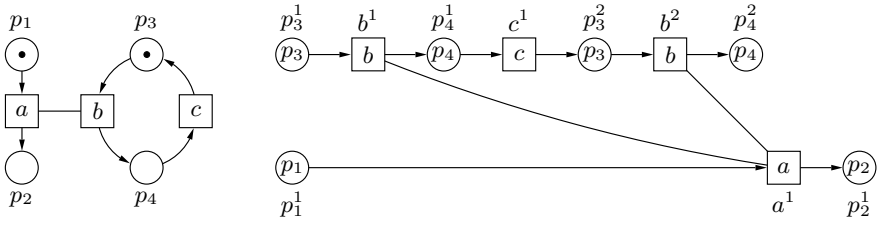


Fig. 8. Mutex arcs may need to connect all potential mutex transitions

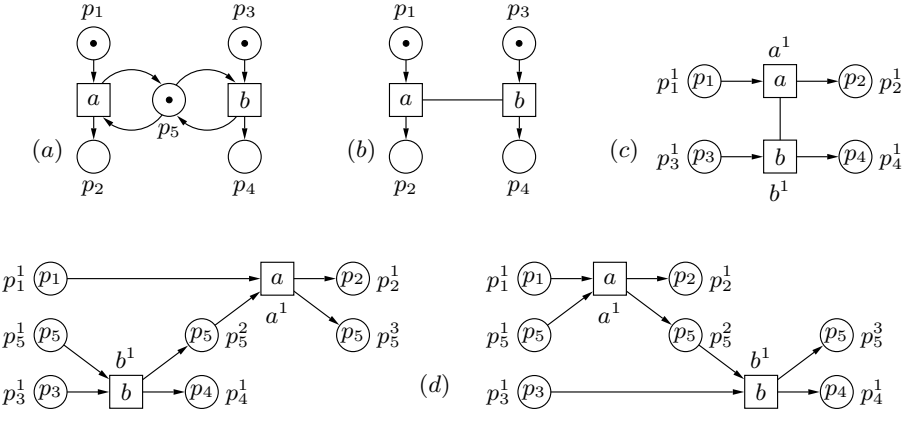


Fig. 9. Mutex arcs lead to more condensed process semantics than self-loops

5 Concluding Remarks

We already mentioned that trying to avoid redundant mutex arcs when constructing processes would require investigation of various paths in the constructed AMO-net. In particular, it would not be sufficient to only consider the most recent transition occurrences. Consider, for example, the ENIM-system shown in Figure 8 and its step sequence $\sigma \stackrel{\text{df}}{=} \{b\}\{c\}\{b\}\{a\}$. The corresponding process, also shown in Figure 8, has two mutex arcs adjacent to the transition a^1. We then observe that dropping the joining of a^1 with b^1 would not be right, as the resulting AMON-net would generate a step sequence $\{a^1, b^1\}\{c^1\}\{b^2\}$, or $\{a, b\}\{c\}\{b\}$ after applying labelling, which is not a valid step sequence of the ENIM-system.

Modelling mutually exclusive transitions can be done in PT-nets using self-loops linking mutually exclusive transitions to a place marked with a single token (which has no other arcs attached to it). This is illustrated in Figure 9(a). An alternative would be to use a mutex arc, as shown in Figure 9(b). At a purely modelling level, there is no real difference between these two representations. However, at the semantical level, the differences can be significant. The point is that mutex arcs represent concurrent histories in a compact way. This should have a direct impact on the size of net unfolding used, in

particular, for model checking. For example, the single process in Figure 9(c) derived for the representation of Figure 9(b) has to be replaced by two processes derived for the representation of Figure 9(a) depicted in Figure 9(d). It is important to observe that these two non-isomorphic processes cannot be equated using the so-called token swapping technique from [1], as the PT-net is 1-safe, suggesting that the potential state space reductions due to mutex arcs have not been considered so far. Intuitively, mutex arc stem from a different philosophy to self-loops. Whereas the latter are related to resource sharing, mutex arcs are derived from semantical considerations and so can provide a more convenient modelling tool.

In our future work we plan to investigate the relationship between mutex arcs and other modelling concepts such as localities [15] and policies [3], also from the point of view of the synthesis of nets where unorderedness does not imply simultaneity of executed actions.

In this paper we did not consider ENM-systems, i.e., EN-systems extended with mutex arcs, as it was our intention to investigate a system model corresponding to the most general paradigm π_1. In future, we intend to find out where ENM-systems fit into the approach presented here.

Acknowledgements

We would like to thank the anonymous reviewers for their suggestions which have led to an improved presentation of our results.

This research was supported by the "Pascal Chair" award from Leiden University, the EPSRC VERDAD project, and NSFC Grants 60910004 and 2010CB328102.

References

1. Best, E., Devillers, R.: Sequential and Concurrent Behaviour in Petri Net Theory. Theoretical Computer Science 55, 87–136 (1987)
2. Billington, J.: Protocol Specification Using P-Graphs, a Technique Based on Coloured Petri Nets. In: Part II of [17], pp. 331–385 (1998)
3. Darondeau, P., Koutny, M., Pietkiewicz-Koutny, M., Yakovlev, A.: Synthesis of Nets with Step Firing Policies. Fundamenta Informaticae 94, 275–303 (2009)
4. Donatelli, S., Franceschinis, G.: Modelling and Analysis of Distributed Software Using GSPNs. In: Part II of [17], pp. 438–476 (1998)
5. Esparza, J., Bruns, G.: Trapping Mutual Exclusion in the Box Calculus. Theoretical Computer Science 153, 95–128 (1996)
6. Gaifman, H., Pratt, V.R.: Partial Order Models of Concurrency and the Computation of Functions. In: LICS, pp. 72–85. IEEE Computer Society, Los Alamitos (1987)
7. Guo, G., Janicki, R.: Modelling concurrent behaviours by commutativity and weak causality relations. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 178–191. Springer, Heidelberg (2002)
8. Janicki, R.: Relational Structures Model of Concurrency. Acta Informatica 45, 279–320 (2008)
9. Janicki, R., Koutny, M.: Structure of Concurrency. Theoretical Computer Science 112, 5–52 (1993)

10. Janicki, R., Koutny, M.: Semantics of Inhibitor Nets. Information and Computation 123, 1–16 (1995)
11. Janicki, R., Koutny, M.: Order Structures and Generalisations of Szpilrajn's Theorem. Acta Informatica 34, 367–388 (1997)
12. Janicki, R., Koutny, M., Kleijn, J.: Quotient Monoids and Concurrent Behaviours. In: Martín-Vide, C. (ed.) Scientific Applications of Language Methods. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory, vol. 2, World Scientific, Singapore (2010)
13. Juhás, G., Lorenz, R., Mauser, S.: Causal Semantics of Algebraic Petri Nets distinguishing Concurrency and Synchronicity. Fundamenta Informaticae 86, 255–298 (2008)
14. Kleijn, H.C.M., Koutny, M.: Process Semantics of General Inhibitor Nets. Information and Computation 190, 18–69 (2004)
15. Kleijn, J., Koutny, M., Rozenberg, G.: Petri Net Semantics for Membrane Systems. Journal of Automata, Languages, and Combinatorics 11, 321–340 (2006)
16. Lengauer, C., Hehner, E.C.R.: A Methodology for Programming with Concurrency: An Informal Presentation. Science of Computer Programming 2, 1–18 (1982)
17. Reisig, W., Rozenberg, G. (eds.): APN 1998. LNCS, vol. 1491 & 1492. Springer, Heidelberg (1998)
18. Rozenberg, G., Engelfriet, J.: Elementary Net Systems. In: Part I of [17], pp. 12–121 (1998)
19. Szpilrajn, E.: Sur l'extension de l'ordre partiel. Fundamenta Mathematicae 16, 386–389 (1930)

On the Origin of Events: Branching Cells as Stubborn Sets*

Henri Hansen[1] and Xu Wang[2]

[1] Department of Software Systems, Tampere University of Technology
P.O. Box 553, FI-33101 Tampere, Finland
henri.hansen@tut.fi
[2] International Institute of Software Technology, United Nations University
Macau, P.O. Box 3058
wx@iist.unu.edu

Abstract. In prime event structures with binary conflicts (pes-bc)[1] a *branching cell* [1] is a subset of events closed under *downward causality* and *immediate conflict* relations. This means that no event outside the branching cell can be *in conflict with* or *enable* any event inside the branching cell. It bears a strong resemblance to *stubborn sets*, a partial order reduction method on transition systems. A stubborn set (at a given state) is a subset of actions such that no execution consisting entirely of actions outside the stubborn set can be in conflict with or enable actions that are inside the stubborn set.

A rigorous study of the relationship between the two ideas, however, is not straightforward due to the facts that 1) stubborn sets utilise sophisticated causality and conflict relations that invalidate the *stability* and *coherence* of event structures [18], 2) without stability it becomes very difficult to define concepts like prefixes and branching cells, which prerequire a clear notion of causality, and 3) it is challenging to devise a technique for identifying 'proper' subsets of transitions as 'events' such that the induced event-based system captures exactly the causality and conflict information needed by stubborn sets.

In this paper we give a solution to the problems by providing an unfolding of labelled transition systems into *configuration structures*, a more general structure supporting or-causality and finite conflict. We show that the branching cell definition can be extended to configuration structures and that each branching cell in the unfolding is a *long-lived* stubborn set, such that no matter how the system evolves, what remains of the branching cell is always a stubborn set.

1 Introduction

Partial Order Reduction (POR) has been successful in alleviating the state explosion problem that manifests in the verification of concurrent systems. Although

* Supported by the PEARL project (041/2007/A3) from FDCT (Macau).

[1] Pes-bc was originally called event structures in [8]. Later Winksel introduced, in [18], a more general structure, which was also named event structure. In this paper, we use the name event structure for the latter.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 248–267, 2011.

drawing ideas and intuition from non-interleaving models of concurrency (esp. Mazurkiewicz traces [7]), most existing POR methods, e.g. the ample, persistent and stubborn set methods [9,3,12] and confluence based methods [4,6], are built on top of interleaving models, mostly transition systems of various types.

On the face of it, it seems straightforward that notions like stubborn sets can be projected onto non-interleaving models and we can readily obtain their true-concurrency counterparts. In particular, branching cells [1,17] of pes-bc bear resemblance to stubborn sets. A closer look of the problem, however, shows that the projection is nontrivial and should better be carried out in two steps.

The first step is fairly easy: in a non-interleaving model it is intuitively simple to define a construct that captures the essence of stubborn sets, which, surprisingly though, is not branching cells. The second step is where the difficulty lies. The correspondence between stubborn sets and the construct cannot be established if we limit our notions of conflict and causality to those of pes-bc, when projecting transition systems into event-based models. It seems stubborn sets do implicitly utilise the extra concurrency affordable by or-causality and finite conflicts (details in Section 3.2). Thus our event-based model have to be more general than pes-bc.

In this paper our projection from transition systems to event-based systems is based on similar frameworks as those of [19,10,14]. We use local 'diamonds' in LTSes and identify an event with the subset of parallel edges (i.e. transitions) in a 'diamond-ful' subgraph. In [19,10], the proposed formalism is *transition systems with indepedence*. However, given a transition system, the consistency conditions (on the independence relations) rule out a significant amount of diamonds to be interpreted as independence. The derived event-based model will have a simplified structure regarding causality and conflicts, and consequently fails to capture as independence some of the diamonds utilised by stubborn sets. In [14], a formalism called *process graphs* is proposed. They can be projected into very general event-based models with more complicated structure regarding causality and conflicts. Indeed, process graphs try to exploit as many diamonds as possible, to the extent that even *resolvable conflicts* [14] can be included in a single (non-interleaving) run of the system. This contradicts with the intuition of stubborn sets, which treats resolvable conflicts as normal conflicts.

In this paper we propose a new technique, a dynamic procedure to identify events from LTSes and build a corresponding event-based system, utilising the same class of diamonds as POR. We use configuration structures [15] as our model of event-based systems. It is a general structure supporting finer grained causality and conflicts needed by stubborn sets. Moreover, compared to [19,10,14], our procedure supports dynamic assignment of independence relations: a pair of transitions can be dependent in one execution while be independent in another.

Our paper is organized as follows. In Section 2 and Section 3, we outline the basic definitions needed, on one hand, for LTSes and stubborn sets and, on the other hand, for configuration structures and branching cells. Then we introduce the notion of *extended knots*, which is defined w.r.t. a given state of the system,

and show that a branching cell corresponds to an extended knot that spans over a given subgraph of a system. Section 4 gives the construction and correctness proof of *unfolding* LTSes into configuration structures and mapping (subsets of) transitions into events. In Section 5 we prove the correspondence result between extended knots and stubborn sets. Section 6 concludes.

2 Definitions and Fundamentals

2.1 Labelled Transition Systems

Definition 1. *A* deterministic labelled transition system *(DLTS) is a 4-tuple* $(S, \Sigma, \Delta, \hat{s})$ *where*

- S *is a set of* states*,*
- Σ *is a finite set of* actions *(ranged over by a, b, etc.)*[2]*,*
- Δ *is a partial function from* $S \times \Sigma$ *to* S *(i.e the* transition *function), and*
- $\hat{s} \in S$ *is the* initial state*.*

For a given DLTS $L = (S, \Sigma, \Delta, \hat{s})$, we define the following:

- $s \xrightarrow{a} s'$ means $(s, a, s') \in \Delta$,
- $s \xrightarrow{a_1 \cdots a_n} s'$ means there exists $s_0, \ldots s_n$ such that $s_0 = s$, $s_n = s'$ and for each $1 \leq i \leq n$, $s_{i-1} \xrightarrow{a_i} s_i$.
- $s \xrightarrow{a}$ means $\exists s' : s \xrightarrow{a} s'$, and $s \not\xrightarrow{a}$ means that $s \xrightarrow{a}$ does not hold;
- $eb(s) = \{a | s \xrightarrow{a}\}$ denotes the set of actions enabled at s.

When there is any danger of confusion, we use $\rightarrow_\Delta$ to say it is derived from a DLTS with Δ as the transition function.

The following definition of stubborn sets can be found in [13], albeit notation may differ slightly.

Definition 2. *Let* $L = (S, \Sigma, \Delta, \hat{s})$ *be a DLTS and let* $s_0 \in S$ *be a state. A set* $T \subseteq \Sigma$ *is* a stubborn set at s_0 *if for every* $a \in T$ *and* $b_1, \ldots, b_k \notin T$ *the following hold:*

1. *If* $s_0 \xrightarrow{a} s'_0$ *and* $s_0 \xrightarrow{b_1 \cdots b_k} s_k$*, then* $s'_0 \xrightarrow{b_1 \cdots b_k} s'_k$ *and* $s_k \xrightarrow{a} s'_k$
2. *if* $s_0 \xrightarrow{b_1 \cdots b_k} s_k \xrightarrow{a}$*, then* $s_0 \xrightarrow{a}$
3. $T \cap eb(s_0) = \{\}$ *if and only if* $eb(s_0) = \{\}$

Furthermore, if additionally $T \subseteq eb(s_0)$ *holds, we say* T *is* a persistent set at s_0.

Unlike, e.g., for persistent/ample sets [9,2], the definition of a stubborn set does not exclude actions that are not enabled (at s_0). For application in state space reduction this makes no real difference. If we are interested only in the enabled subset of actions of a stubborn set T, we denote it by $T \cap eb(s_0)$, which is exactly a persistent set.

It is perhaps easier to understand the stubborn set definition in terms of five types of *state-local* binary relations between actions for a given LTS, $(S, \Sigma, \Delta, \hat{s})$. Any local relation between say action a and b w.r.t. a state s_0 can be calculated by checking at most four transitions 'local' to s_0: the a and b transitions at s_0, the b transition after a, and the a transition after b (c.f. Figure 1).

[2] We further assume that, for each $a \in \Sigma$, there is a reachable state s such that $s \xrightarrow{a}$.

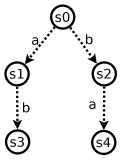


Fig. 1. Local transitions at s_0

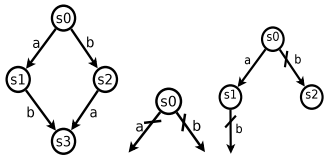


Fig. 2. Conditional independence $(a||_{s_0}b)$

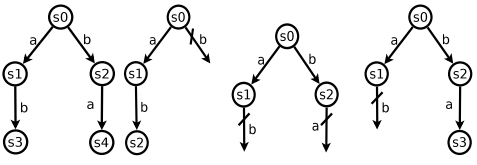


Fig. 3. Shapes for (the four types of) non-independence

With the four potential transitions, we can construct a number of possible 'shapes' for local transition systems at s_0. The set of shapes can be partitioned into five groups; each one characterised by a type of binary relations between a and b:

Definition 3. *Let $a, b \in \Sigma$ be actions of a DLTS and $s_0 \in S$ be a state.*

- *a and b are* conditionally independent *at s_0 (i.e. $a||_{s_0}b$) if one of the following hold (c.f. Figure 2):*
  1. $s_0 \xrightarrow{ab} s_3$, $s_0 \xrightarrow{ba} s_4$, *and* $s_3 = s_4$,
  2. $s_0 \xrightarrow{a}$ *and* $s_0 \not\xrightarrow{b}$, *or*
  3. $s_0 \xrightarrow{a} s_1$, $s_0 \not\xrightarrow{b}$ *and* $s_1 \not\xrightarrow{b}$ *(or symmetrically by exchanging a and b).*
- *a and b are* non-commutative *if $s_0 \xrightarrow{ab} s_3$, $s_0 \xrightarrow{ba} s_4$, and $s_3 \neq s_4$ (c.f. the first case in Figure 3).*
- *a* enables *b if $s_0 \not\xrightarrow{b}$ and $s_0 \xrightarrow{ab}$ (c.f. the second case in Figure 3).*
- *a and b* symmetrically disable *each other if $s_0 \xrightarrow{a}$ and $s_0 \xrightarrow{b}$ but $s_0 \not\xrightarrow{ab}$ and $s_0 \not\xrightarrow{ba}$ (c.f. the third case in Figure 3).*
- *a* asymmetrically disables *b if $s_0 \xrightarrow{a}$ and $s_0 \xrightarrow{ba}$, but $s_0 \not\xrightarrow{ab}$ (c.f. the fourth case in Figure 3).*

Note that conditional independence, non-commutativity, and symmetrical disabling are irreflexive symmetric relations while enabling and asymmetrical disabling are irreflexive antisymmetric relations. Furthermore we say a and b are *in conflict* at s_0 if either a and b are non-commutative, or a and b symmetrically or asymmetrically disable each other. Conflict is an irreflexive symmetric relation.

Using the five binary relations, a non-local property (w.r.t. a state) can sometimes be decomposed into a series of local relations over a subgraph of states[3].

[3] Decomposability, in practice, implies the existence of efficient algorithms to analyse such non-local properties.

For example, stubborn sets are characterized by these local relations according to the following folk theorem.

Proposition 1. *The non-empty set $T \subseteq \Sigma$ is a stubborn set at $s_0 \in S$ iff, for every $a \in T$ and $b \notin T$,* ***either*** *a* ***and*** *b* ***are conditionally independent at*** *s_k* ***or*** *a* ***enables*** *b* ***at*** *s_k (i.e. neither a and b are in conflict at s_k nor b enables a at s_k) for all states s_k reachable from s_0 by using non-T transitions, i.e. $s_0 \xrightarrow{b_1 \cdots b_k} s_k$ with $b_1, \ldots, b_k \in \Sigma \setminus T$.*

Proof. From SS (stubborn set) to local definition: Assume that neither a and b are conditionally independent nor a enables b at s_k, and that s_k is the first such state on the path $s_0 \xrightarrow{b_1 \cdots b_n} s_k$. Then there are three possibilities: 1) a is enabled at s_0 and a and b symmetrically or asymmetrically disable each other at s_k, 2) a is not enabled at s_0 and b enables a at s_k, and 3) a is enabled at s_0 and a and b are non-commutative at s_k. The second case contradicts the second requirement of stubborn sets; while the other two cases contradict the first requirement.

From local definition to SS: Let us assume $s_0 \xrightarrow{a} s'$ and $s_0 \xrightarrow{b_1 \cdots b_k} s_k$. It is clear that a and b_i are conditionally independent at each state s_{i-1} (such that $s_0 \xrightarrow{b_1 \cdots b_{i-1}} s_{i-1}$) for every $1 \leq i \leq k$; it cannot be true that a enables b_i at s_{i-1} since b_i is already enabled at s_{i-1}. Therefore we know that $s_i \xrightarrow{a}$. From conditional independence it also follows that $s_{i-1} \xrightarrow{a} s'_{i-1} \xrightarrow{b_i} s'_i$ and $s_i \xrightarrow{a} s'_i$, for each $i \leq k$, and the first requirement of stubborn sets is satisfied.

Assume then, that $s_0 \xrightarrow{b_1 \cdots b_k} s_k \xrightarrow{a}$. Now, the independence/enabling requirement states that none of the b_i for $1 \leq i \leq k$ enables a, so the second requirement readily follows, and T must be stubborn.

3 Configuration Structures and Branching Cells

In this section we first introduce configuration structures [16,15]. Then, as our original contribution, we develop the new notion of *knots* and *extended knots*, which are the non-interleaving counterparts of persistent sets and stubborn sets resp. (see proofs in Section 5). Finally, we extend to configuration structures the definition of *prefixes*, *immediate conflicts* and *branching cells*, and show how branching cells are actually long-lived extended knots.

3.1 Well-Formed Configuration Structures

Definition 4. *A* configuration structure *(or simply CS) is a pair (E, C), where*

- *E is a potentially infinite set of* events*, and*
- *C is a set of* configurations *over E, where a configuration $c \in C$ is a finite subset of E.*

A configuration c can be thought of as representing a state reached after an execution of exactly the set c of events. The empty configuration $\{\}$ represents the initial state and is a required member of C in our model.

Below we fix a configuration structure $cs = (E, C)$ and introduce some basic notions (based on those in [16,15]) for CSes.

- We say there is a *step-transition* from c to c', written $c \xrightarrow{c' \setminus c}_C$, if $c \subseteq x \subseteq c' \implies x \in C$ holds for all $x \subseteq E$. (Note that in this paper we do not list explicitly the destination configuration of the transition, i.e. c'.)
- Sometimes we use $c \xrightarrow{e}_C$ to mean $c \xrightarrow{\{e\}}_C$, i.e. e is *enabled* at c. We write $eb(c) = \{e \in E \mid c \xrightarrow{e}_C\}$, to denote the set of enabled events at c.
- We say cs is *finitely branching* if $eb(c)$ is finite for all $c \in C$. In such a cs *infinite configurations* can be derived from finite ones and there is no unbounded concurrency and conflict.
- For a nonsingleton set x, $c \xrightarrow{x}_C$ represents the concurrent execution of multiple events. Obviously, a nonsingleton step-transition is reducible to a sequence of singleton step-transitions.
- We say a nonempty finite set $K \subseteq E$ is a *consistent set* if there is $c \in C$ such that $K \subseteq c$. Otherwise, K is an *inconsistent set*.
- We say that cs is *closed under (nonempty) bounded union* if, for all nonempty finite subsets $D \subseteq C$, $\bigcup D$ is a consistent set implies $\bigcup D \in C$.
- We say that cs is *connected* if all the configurations in C are reachable from $\{\}$ by step-transitions, and that cs is *irredundant* if every event from E is contained inside at least one configuration from C.

Based on the above, we can say cs is *well-formed* if it is irredundant, finitely branching, connected and closed under bounded union. Well-formed CSes have roughly the same expressiveness as event structure from [18]. We prefer to use configuration structures in this paper mainly because of its affinity to LTSes. We establish a few basic properties of well-formed CSes:

Lemma 1. *If there is a step-transition path from c to c' such that $e \in eb(c)$ and $c' \cup \{e\}$ is consistent, then we have either $e \in c'$ or $e \in eb(c')$.*

Proof. If c_i and c_{i+1} are two adjacent configurations (connected by event $e' \neq e$) on the path and e is enabled at c_i, closure under bounded union gives us that e is enabled at c_{i+1}.

Lemma 2. *Let $c, c' \in C$ such that $c \subseteq c'$. Then there is a (step-transition) path from c to c'.*

Proof. Follows from Lemma 5.2.5 in [16].

For the rest of this paper we only consider well-formed CSes and simply call them CSes. Furthermore, to gain a deeper understanding of CSes, we need to use a few more notions:

- We say e is *initially enabled* at $c \in C$ (or c is the *cause* of e) if e is enabled at c but not enabled at any $c' \subset c$.

- We use $IE(e)$ to denote the set of configurations at which e is initially enabled. In other words, $IE(e)$ is the set of minimal configurations on which e is enabled.
- We say c has previously *encountered* an event e in its evolution if $e \in A(c) = \{e | \exists c_0 \in IE(e) : c_0 \subseteq c\}$.
- Given a configuration $c \in C$, the *future* of cs after c, is defined to be the configuration structure: $cs/c = (E_{cs/c}, C_{cs/c})$, where $C_{cs/c} = \{c' \setminus c \mid c' \in C \wedge c \subseteq c'\}$ and $E_{cs/c} = \bigcup C_{cs/c}$. Obviously cs/c is well-formed if cs is well-formed.
- We say a step-transition $c \xrightarrow{e}_C$ *disables* an event e' if e' is enabled at c but not at $c \cup \{e\}$.
- We say a step-transition $c \xrightarrow{e}_C$ *eliminates* an event e' from the system if $e' \in E_{cs/c}$ but $e' \notin E_{cs/(c \cup \{e\})}$. According to Lemma 1 and 2, we know that once e' is disabled it is eliminated from the system.

The causes of an event in CSes are not necessarily unique. The loss of uniqueness is either due to *or-causality* or due to *the loss of conflict heredity*. In Figure 4, the 1st and 3rd graphs describe the or-causality case [5,20], i.e. $e3$ or-causally depends on $e1$ and $e2$; while the 2nd and 4th graphs describe the loss-of-conflict-heredity case, i.e. $e3$ causally depends on a pair of conflicting events, i.e. $e1$ and $e2$.[4] The 3rd and 4th graphs are the CSes of the examples, from which we can see that $\{e1\}$ and $\{e2\}$ are both causes of $e3$. The 1st and 2nd graphs are the *'extended event structures'* of the examples[5]: head-conjoined arrows (marked by *or*) represent or-causal dependency while dotted polygons represent conflict relations (i.e. the set of events inside a polygon form a conflict). Furthermore, with 'extended event structure' notation derivable conflicts or causal dependency do not need to be drawn explicitly.

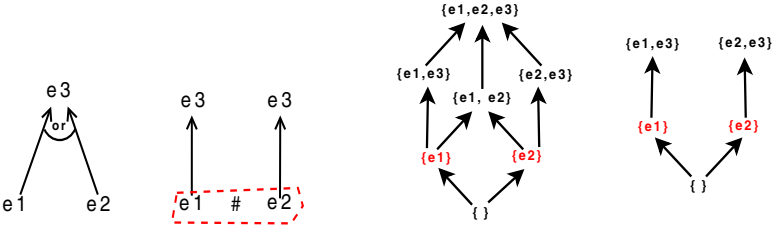


Fig. 4. Two types of multi-causes

[4] Note that the second case will be impossible if we have conflict heredity, from which we can derive $e3$ is in conflict with $e3$, contradicting the irreflexivity of conflict relations.

[5] It might look unlikely that the pathological case of the 2nd graph can produce a well-formed CS. But by checking the 4th graph readers can verify that it is so indeed.

3.2 Knots and Extended Knots

A concurrent system consists of a set, say H, of behaviours. In a non-interleaving setting, such a behaviour, say $h \in H$, is a non-interleaving maximal run of the system, which usually is charaterised by a maximal configuration. A graph traversal algorithm is supposed to exhaustively explore the set of H. The idea of POR lies in that, given any $h \in H$, it is necessary and only necessary to explore one linearisation of h. To this end the following notion of *weak knots* is useful.

- A non-empty subset $K_0 \subseteq eb(\{\})$ is a *weak knot* if, for all $c \in C$, there exists some $e \in K_0$ such that $c \cup \{e\}$ is consistent.

The definition says that, no matter how the system evolves, the resultant configuration either contains a member of K_0 or is extendable to one containing a member of K_0. Thus, ignoring events outside K_0 at the current configuration will not cause the exploration missing any behaviour of the system.

Weak knots seem to be the weakest possible notion preserving behaviours. In this sense the notion has some canonicity and is of independent interests. However, weak knots are too weak to capture persistent sets and stubborn sets.

Persistent sets and stubborn sets impose stronger restrictions. For weak knots, we require that, no matter how the system evolves, at least one member of K_0 has survived (i.e. executed) or will survive (i.e. not eliminated yet), since the evolution of the system may eliminate members of K_0. For persistent sets and stubborn sets, we require that before a members of K_0 has survived no members of K_0 can be eliminated.

- A non-empty subset $K_0 \subseteq eb(\{\})$ is a *knot* if, for all $c \in C$, either $c \cap K_0 \neq \{\}$ or $c \cup \{e\}$ is consistent for all $e \in K_0$.

Thus, knots should be the counterparts to persistent set. The counterpart of stubborn sets is more complicated, which needs to further accommodate the requirement that any member of K_1 that is not enabled yet cannot become enabled without first executing a member of K_1:

- Given any $K \subseteq E$, we say K *can block* an event $e \in E$ if $e \notin K$ and $K \cap c \neq \{\}$ for all $c \in IE(e)$. And we say e can block e' if $\{e\}$ can block e'.
- A non-empty subset $K_1 \subseteq E$ is an *extended knot* if, $K_0 = K_1 \cap eb(\{\})$ is a knot and K_0 can block all $e \in K_1 \setminus eb(\{\})$.

To formally establish the correspondence between knots and persistent/stub -born sets, we need to project LTSes into event-based systems that support or-causality and finite conflicts. The reasons are based on the following observations:

- For a confluent LTS, such as the one derived from the 3rd graph in Figure 4, singleton sets suffice as stubborn sets at each state, which implies the corresponding event-based system consists of a single behaviour, i.e. a conflict-free system. However, without or-causality, there is no way to project the LTS into a conflict-free event-based system.

– With finite conflicts the LTS like the one in Figure 9 can be projected into an event-based system that consists of three non-interleaving runs. This matches with the three sequential runs generated by minimal stubborn sets. However, without finite conflicts the projected event-based system will contain at least four non-interleaving runs.

We will formally define the projection to CSes in Section 5 . But before that, let us first extend the definition of branching cells onto CSes.

3.3 Prefixes, Immediate Conflicts and Branching Cells

The definition of configuration structures is very 'low-level'. We have to build up other notions of true concurrency (e.g. conflict and causality) step by step from configurations. We will see that some of the notions are surprisingly subtle to define and there has been few previous works we can rely on.

– We say an inconsistent subset $K \subseteq E$ is a *conflict set* (at $\{\}$) if all its strict subsets are consistent. We say cs is *conflict-free* if cs has no conflict set. Obviously cs is conflict-free iff $E \in C$.

The arity or degree of conflicts is not fixed in configuration structures: a conflict may involve any number of events. Thus we use *conflict sets* (rather than conflict tuples) to define conflicts. A conflict set is a dynamic notion relative to (i.e. indexed by) a configuration: as cs evolves, conflicts in cs evolve too. For instance, an initially consistent subset $K \subseteq E$ may become a conflict later on (say in cs/c). On the other hand, conflict-freedom is preserved by the evolutions of cs. Let us take a closer look at conflict dynamism.

We say cs/c *inherits* conflict K of cs if $K \setminus c$ is a conflict of cs/c. Actually, all the conflict sets in cs/c are inherited from cs:

Lemma 3. *Given $e \in eb(\{\})$, K is a conflict of $cs/\{e\}$ implies K or $K \cup \{e\}$ is a conflict of cs.*

Proof. If K is inconsistent in cs, then obviously K is a conflict set of cs. If K is consistent in cs, then $K \cup \{e\}$ is a conflict set of cs since $K \cup \{e\}$ is inconsistent and $(K \setminus \{e'\}) \cup \{e\}$ is consistent in cs for all $e' \in K$.

It is also possible that, with cs evolution, some conflicts of cs will be *discarded* by cs/c. An example is that, given a conflict K of cs, suppose an enabled event $e \notin K$ is in a ternary conflict with $e', e'' \in K$[6]. By executing e, the ternary conflict is reduced to conflict $\{e', e''\}$ in $cs/\{e\}$, making K obsolete in $cs/\{e\}$.

Proposition 2. *Given a conflict K of cs and $e \in eb(\{\})$,*

1. *$K = \{e, e'\}$ implies e' is eliminated in $cs/\{e\}$, i.e. $e' \notin E_{cs/\{e\}}$,*

[6] That is, $(E = \{e, e', e'', e3\}, C = \{c \subseteq E | \{e, e', e''\} \not\subseteq c \wedge K \not\subseteq c\})$, where $K = \{e', e'', e3\}$.

2. *$e \in K$ and K is not binary implies that $K \setminus \{e\}$ is a conflict of $cs/\{e\}$ (i.e. indirect inheritance),*
3. *$e \notin K$ but there exists another conflict K' with $e \in K'$ and $K' \setminus \{e\} \subseteq K$ implies K is not a conflict in $cs/\{e\}$ (i.e. discard), and*
4. *$e \notin K$ and there does not exist any conflict K' with $e \in K'$ and $K' \setminus \{e\} \subseteq K$ implies K is a conflict in $cs/\{e\}$ (i.e. direct inheritance).*

Proof. Case 1 is obvious. For case 2, it comes from the facts that $K \setminus \{e\}$ is inconsistent in $cs/\{e\}$ but $K \setminus \{e'\}$ is consistent in $cs/\{e\}$ for all $e' \in K \setminus \{e\}$. Case 3 is also straightforward after separating binary K' from non-binary K'. For case 4, we can first prove that $K \subseteq E_{cs/\{e\}}$ since otherwise a binary K' satisfying the conditions must exist in order to eliminate a member of K. Then, since K is inconsistent in $cs/\{e\}$ and no subset of K can be a conflict of $cs/\{e\}$ (due to Lemma 3), it concludes that K is a conflict of $cs/\{e\}$.

The notion of causality is even more subtle. An extreme example of such subtlety can be found in Figure 5. At $\{e1\}$, the execution of $e2$ enables $e4$; while at $\{e3\}$, the execution of $e4$ enables $e2$. Thus $e2$ and $e4$ (or-)causally depend on each other. The 'extended event structure' representation of the same example can be found in Figure 6 (the left graph).

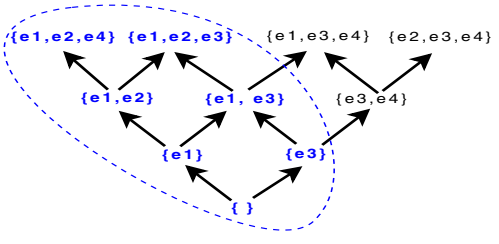


Fig. 5. Cyclic (or-)causality

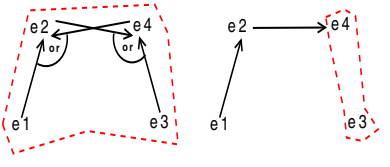


Fig. 6. Two example prefixes

In this paper we will not try to give an explicit definition to causality. Rather, we define notions that involve causality only implicitly.

Given a finite $D \subseteq C$, we say D is *downward-closed* (w.r.t. $\subseteq$) if $c \subseteq c' \in D \implies c \in D$, and D is *bounded-union closed* if $c, c' \subseteq D \wedge c \cup c' \in C \implies c \cup c' \in D$. We use $D^{\downarrow}$ to mean the downward closure of D. Then,

- We say a finite subset of configurations $D \subseteq C$ is a *prefix* if there exists $D_0 \subseteq C$ so that D_0 is downward-closed and D is the bounded-union closure of D_0. Given a prefix D', another prefix D is a *sub-prefix* of D' if $D \subseteq D'$. Prefixes induce well-formed CSes, i.e. $(\bigcup D, D)$.

Note that prefixes are subsets of configurations rather than subsets of events since the latter is not expressive enough[7]: In Figure 5 both the whole graph and

[7] Obviously with event-based prefixes it becomes impossible (due to or-causality) to extend a prefix on an event by event basis, if prefixes are still interpreted as 'downward-closure of causality'.

the subgraph carved out by the dotted ellipse are prefixes (corresponding to the left and right 'extended event structures' in Figure 6 resp.) of the CS; but their sets of involved events are the same, i.e. $\{e_1, e_2, e_3, e_4\}$.

Note also that prefixes are not necessarily downward-closed. In Figure 7, the bounded-union closure of $\{\{e,d,c\},\{e,b\},\{a\}\}^{\downarrow}$ is not downward closed.

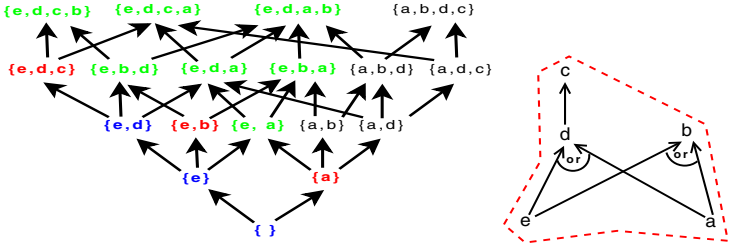


Fig. 7. A more sophisticated prefix

– A prefix D' is *convex* if, for all other prefix D with $\bigcup D = \bigcup D'$, we have $D \subset D'$.

Given $B \subseteq E$, the *projection* of cs onto B, $cs{\restriction_B} = (E_{cs{\restriction_B}}, C_{cs{\restriction_B}})$, is defined to be $C_{cs{\restriction_B}} = \{c \mid c \in C \wedge c \subseteq B\}$ and $E_{cs{\restriction_B}} = \bigcup C_{cs{\restriction_B}}$. For a convex prefix D it is completely characterised by $\bigcup D$ since $D = cs{\restriction_{\bigcup D}}$.

We will use D and $\bigcup D$ interchangeably for convex prefixes, and we have, for any two convex prefixes B and B', B is a sub-prefix of B' iff $B \subseteq B'$.

An interesting subclass of convex prefixes are the so-called *strong prefixes*.

– A prefix D is a *s-prefix* if, for all $e \in \bigcup D$ and $c \in IE(e)$, we have $c \in D$.

The alternative definition of s-prefix is: $B \subset E$ is a *s-prefix* if, for all $e \in B$ and $c \in IE(e)$, $c \subseteq B$. Note the distinctions between prefixes and s-prefixes: for each $e \in \bigcup D$ a prefix requires inclusion of one cause of e while an s-prefix requires inclusion of all causes of e.

Now we are ready to define immediate conflicts and branching cells.

– We say a subset of events $K \subseteq E$ is a *local conflict set* (LCS) at c if $K \subseteq eb(c)$ and K is a conflict of cs/c. Given two LCSes K at c and K' at c', we say K' at c' is *derivable from* K at c if $K' \subseteq K$ and $c \subset c'$ (c.f. direct and indirect inheritance of conflicts).
– We say a subset of events $K \subseteq E$ is an *immediate conflict set* (ICS) at c if K at c is an LCS that is not derivable from any other LCS.
– We say a prefix D *characterises* an ICS (i.e. $\bigcup D \setminus \bigcap max(D)$ at $\bigcap max(D)$, where $max(D)$ denotes the set of maximal elements of D) if D is a minimal prefix that is not conflict-free[8].

[8] A prefix D is *conflict-free* if the CS it induces is conflict-free.

The first definition of ICS captures the localness of immediate conflicts. For instance, in Figure 7, $\{a, b, c\}$ at $\{e, d\}$ is an ICS but $\{a, b\}$ at $\{e, d, c\}$ is not an LCS. The second definition, i.e. characterisation, lifts the orginal definition on pes-bc. We can see that the two definitions coincide:

Lemma 4. *A prefix D characterises an immediate conflict implies that $\bigcup D \setminus \bigcap max(D)$ is an ICS at $\bigcap max(D)$; and K is an ICS at c implies there is a prefix characterising it.*

Proof. From characterisation to ICS: For any $c \in max(D)$, find some $e \in \bigcup D \setminus c$ and $c_0 \in IE(e)$ such that $c_0 \subseteq c$. Then the bounded-union closure of $\{c, c_0 \cup \{e\}\}^{\downarrow}$ gives rise to D since there is no prefix that is (strictly) larger than $\{c\}^{\downarrow}$ but smaller than D. Thus $c \cup \{e\} = \bigcup D$. Furthermore, for any $c' \in D$ with $c' \setminus c = \{e\}$, there exists some $c_1 \in \{c\}^{\downarrow}$ such that $c_1 \cup (c_0 \cup \{e\}) = c'$. Thus we have $c_0 \cup c_1 = c \cap c' \in C$. Using this rule we can intersect members of $max(D)$ up one by one. Eventually we get $\bigcap max(D) \in C$ and $\bigcup D \setminus \bigcap max(D)$ is an LCS at $\bigcap max(D)$. Also we can easily see that if the LCS is derivable from another LCS. Then D cannot be minimal.

From ICS to characterisation: Let D be a minimal prefix that contains $M = \{c \cup (K \setminus \{e\}) | e \in K\}$. K at c being ICS implies $c \in IE(K)$. The minimality of D implies that, for all $e \in K \cup c$ and $c_0 \in D$, $c_0 \in IE(e)$ implies $c_0 \subseteq c$. Thus there is no $c' \in D$ such that $K \subseteq c'$ and we have $max(D) = M$. Since adding $K \cap c$ to D will give rise to a confluent CS, say cs', and any strict sub-prefix of D is also a strict sub-prefixes of $D \cup \{K \cap c\}$, all strict sub-prefix of D are conflict-free.

It seems that immediate conflict and initial enabledness are two fundamental notions in CSes. They together completely characterise CSes.

Lemma 5. *Two CSes $cs = (E, C)$ and $cs' = (E', C')$ are isomorphic iff there is a bijection ϕ from E to E' such that 1) $\phi(IE_{cs}(e)) = IE_{cs'}(\phi(e))$ for all $e \in E$ and 2) K at c is an ICS of cs iff $\phi(K)$ at $\phi(c)$ is an ICS of cs'.*

Proof. Use the fact that $c \in C$ implies there is (e_1, c_1), (e_2, c_2), ..., (e_n, c_n) such that $c_i \in IE(e_i)$ for all $1 \leq i \leq n$ and $c = \bigcup_{1 \leq i \leq n} c_i \cup \{e_i\}$.

Based on immediate conflicts we can define branching cells:

- A s-prefix B is an (initial) *branching cell*[9] if it is a minimal non-empty s-prefix such that, for all $c \in C$, K is an ICS at c and $K \cap B \neq \{\}$ implies $K \subseteq B$.

Note that we use s-prefix to define branching cells. That is because, if not all causes are included in a branching cell, it might allow the possibility that outside events enable inside events, contradicting the requirements of stubborn sets. Given a branching cell B and the sub-structure $cs \restriction_B$ carved out by B, the evolution of $cs \restriction_B$ is independent to the evolution of the rest of the system:

[9] On pes-bc a subset B of events is called an initial (i.e. at configuration $\{\}$) branching cell, or an initial stopping prefix, if it is a minimal non-empty subset closed under both immediate conflict and downward causality.

Lemma 6. *Given a branching cell B of cs, $c \cap B \in C$ and $(cs/c)\restriction_B = (cs/(c \cap B))\restriction_B$ hold for all $c \in C$.*

Proof. Easy to see $c \cap B \in C$ as, for all $e \in c \cap B$, there is $c^e_B \in IE(e)$ such that $c^e_B \subseteq c$. Obviously $c \cap B = \bigcup_{e \in c \cap B}(c^e_B \cup \{e\}) \in C$ (due to bounded union closure). Assume $(cs/c)\restriction_B = (cs/(c \cap B))\restriction_B$ is not true. Then there exists $c \in C$, $c_B \subseteq B$ and $e \in E \setminus B$ such that $c_B \in C_{cs/c}$, $c \xrightarrow{e}_C c'$ and $c_B \notin C_{cs/c'}$. Thus e must be involved in a conflict (of cs/c) with a subset of c_B. Then on a (singleton) path from c to $c \cup c_B$, e will eventually be eliminatesed by one of its transition $c'' \xrightarrow{e'}$. Obviously, at c'', $\{e, e'\}$ is an LCS. Contradiction with BC (branching cell) definition!

3.4 Knots and Extended Knots are Branching Cells

Branching cells are defined in terms of immediate conflicts, whereas knots are in terms of event elimination. The two ideas, however, are closely linked since any ICS containing events from both inside and outside of a knot can evolve into a binary conflict on a pair of inside and outside events that eliminate each other.

The evolution considered in the branching cell definition is based on events from both inside and outside, whereas that in knot definition is based on outside events only. As a consequence, knots is a local (i.e. dynamic) notion whereas branching cell is a global notion.

Branching cells can contain events that are not yet enabled, whereas knots contain only enabled events. In this sense, knots is less closely linked to branching cells than extended knots is. The downward-causality closure condition (i.e s-prefix) of branching cells appears in extended knots partially as the condition that the enabled subset of inside events can block the rest of inside events (i.e. those not yet enabled).

Theorem 1. *Given a configuration structure $cs = (E, C)$, $B \subseteq E$ is a branching cell of cs iff B is a minimal non-empty s-prefix of cs such that $eb(c) \cap B$, if non-empty, forms a knot of cs/c for all $c \in C$.*

Proof. We first prove that, given any s-prefix B, ICS closure and long-lived knot are equivalent. Then, automatically the minimality of one leads to the minimality of the other.

From ICS closure to long-lived knot: $K_0 = eb(\{\}) \cap B$ is a knot at $\{\}$ since 1) eliminating any member of K_0 requires a binary LCS involving e and a non-B event at some subsequent future reachable by non-B events, 2) mixed binary LCSes implies mixed ICSes and thus contradiction with BC definition! $eb(c) \cap B$ is a knot at $c \in C$ since what remains of B, or the *residues* of B, continues to be branching cells.

From long-lived knot to ICS closure: Assume there is a mixed ICS K at $c \in C$. Select one event e from $K \cap B$ and one event e' from $K \setminus B$. Then there exists $c \subseteq c' \in C'$ such that $\{e, e'\}$ is an LCS at c'. Obviously $eb(c') \cap B$ is neither empty nor a knot of cs/c'. Contradiction!

The above theorem works for knots only when B is assumed to be s-prefix. This restriction, however, can be lifted for the case of extended knots.

Corollary 1. *Given a configuration structure $cs = (E, C)$, $B \subseteq E$ is a branching cell of cs iff B is a minimal non-empty subset of E such that $B \cap E_{cs/c}$ (i.e. the residue of B in cs/c), if non-empty, is an extended knot of cs/c for all $c \in C$.*

Proof. Only need to prove that, given any $B \subseteq E$, long-lived knot are equivalent to BC (i.e ICS closure plus s-prefix).

From BC to long-lived ex-knot: Obvious since $B \cap eb(\{\})$ is a knot and can block $B \setminus eb(\{\})$.

From long-lived ex-knot to BC: If B is not an s-prefix, then we can always find $e \in B$ and $c \in IE(e)$ such that $c \setminus B \neq \{\}$ and $\forall e' \in B \cap c : c_0 \in IE(e') \wedge c_0 \subseteq c \implies c' \subseteq B$. Consequently $c \cap B \in C$ holds and $B \setminus c$ is not an ex-knot of $cs/(c \cap B)$. Contraditions! ICS closure is obvious.

4 Events in LTSes

We now establish a connection between configuration structures and LTSes. Our method follows the tradition of identifying *events* (of a non-interleaving model) with *subsets of transitions* (of a transition system). Given such a subset, the transitions inside it are the occurrences of the same event under different global states. They form a set of parallel edges on a 'diamond-ful' subgraph of the transition system. However, unlike previous works which are more interested in the direction from event-based systems to transition systems (e.g. characterising the subclass of transition systems produced by a given type of event-based systems, say one-safe PN), our work focuses more on the direction from transition systems to event-based systems and we use the full class of DLTSes.

Definition 5. *Let $L = (S, \Sigma, \Delta, \hat{s})$ be a DLTS. A* labelled CS *over L is a tuple (E, C, lb, st), where*

- *(E, C) constitutes a CS*
- *$lb : E \to \Sigma$ is a function*
- *$st : C \to S$ is a function*

A labelled CS $lcs = (E, C, lb, st)$ over L is an *unfolding* of L, if

(a) $st(\{\}) = \hat{s}$,
(b) for all $c \in C$, lb maps the set of enabled events at c (i.e. $eb_C(c)$) in an 1-to-1 fashion to the set of enabled actions at $st(c)$ (i.e. $eb_\Delta(st(c))$),
(c) for all $c \in C$, $c \xrightarrow{e}_C c \cup \{e\}$ implies $st(c) \xrightarrow{lb(e)}_\Delta st(c \cup \{e\})$, and
(d) Given any $c' \in C$ and any $e' \in A(c')$ (i.e. one of its previously encountered event), we have $e' \in eb(c')$ iff for all transitions $c \xrightarrow{e}_C c \cup \{e\}$ with $c \cup \{e\} \subseteq c'$, $e \neq e'$ and $lb(e)$ and $lb(e')$ are not in conflict at $st(c)$.

The four conditions in the unfolding definition seem to be natural requirements. The last one essentially says that an enabled event e' at c' should not be in conflict with any $e \in c'$ (i.e. judging from their DLTS mappings). It can be equivalently reformulated as: Given any $c' \in C$ and $e' \in A(c')$, $e' \in eb(c')$ iff, either $c' \in IE(e')$ or, for all $c \xrightarrow{e}_C c'$ with $e' \in A(c)$, event e' is *transferred* on the transition from c to c', i.e. $e' \in eb(c)$ and $lb(e)||_{st(c)}lb(e')$ (which implies $e \neq e'$). Thus we say c' *inherits* e' if e' is transferred on all $c \xrightarrow{e}_C c'$ with $e' \in A(c)$.

We can now give a procedure $lcs = \mathcal{U}(L)$ to unfold a DLTS (i.e. L) into a labelled CS over L (i.e. lcs).

$\mathcal{U}(L = (S, \Sigma, \Delta, \hat{s}))$

```
1  Set i = 0 and lcs = (E, C, lb, st) = ({}, {{}}, {}, {({}, ŝ)});
2  Set H = {({}, {})};
   -- function H(c) gives the set of inherited events at c
3  While there exists c ∈ C with |c| = i do
4     Foreach c ∈ C with |c| = i do
5        Foreach transition st(c) -a->_Δ s' do
6           If there is e ∈ H(c) such that lb(e) = a Then select e
7           Else generate a new event e, add e to E and (e, a) to lb, and select e;
8           Add c ∪ {e} to C and (c ∪ {e}, s') to st;
9     Foreach c' ∈ C with |c'| = i + 1 do
10       Add (c', {}) to H;
11       Foreach e' ∈ E \ c' such that e' ∈ A(c') do
         -- function IE(e), similarly eb(c), can be derived from (E,C)
         -- even though the current (E,C) might be just a partial CS
12          If ∃c -e->_C c' : e' ∈ A(c) ∧ ¬(e' ∈ eb(c) ∧ lb(e)||_st(c) lb(e'))
13          Then continue Else add e' to H(c');
14       Foreach a ∈ eb(st(c')) -- or-causality event merge
15          If |A = {e : H(c')|lb(e) = a}| > 1 Then substitute a new event e''
16             for every member of A in lcs and H;
17    i := i+1;
18 return (E, C, lb, st)
```

Fig. 8. An unfolding procedure

We can illustrate the procedure by unfolding a broken cube in Figure 9. In the broken cube parallel edges are labelled by the same action; that is, along the three axes, they are resp. a, b and c.

In the step 0 the initial state is mapped to configuration $\{\}$, on which the set of inherited events $H(\{\})$ is empty. Thus it is labelled as $\{\}/\{\}$ (i.e. $c/H(c)$). Since there is no inherited events to cover (i.e. through sharing of labels) any of the outgoing transitions of the initial state, three new events $e1$, $e2$ and $e3$ are generated (note the use of symbol !) for the three outgoing transitions (c.f. line 7 of the code). Each transition (say the one labelled by $e1$) leads to a new configuration (i.e. $\{e1\}$), carrying the set of transferred events (i.e. $\{e2, e3\}$). For instance $e2$ is transferred on the transition $\{\} \xrightarrow{e1}_C$ because the a and b

transitions at the initial state (corresponding to $e1$ and $e2$ resp.) forms a local diamond. Similarly the transfer of $e3$ by $\{\}$ $\xrightarrow{e1}_C$ is due to the a and c diamond at the initial state.

In the step 1, configuration $\{e1\}$ inherits $\{e2, e3\}$ since it is what is transferred on its only incoming transition. And no new event will be generated at $\{e1\}$ since the labels from $\{e2, e3\}$ are sufficient to cover those of the (two) outgoing transitions at $st(\{e1\})$. At configurations $\{e2\}$ and $\{e3\}$, they similarly inherit two events. However, in further exploration of the graph from $st(\{e2\})$ and $st(\{e3\})$ both encounter non-commutativity for their (pairs of) outgoing transitions. Thus $\{e2\}$ $\xrightarrow{e3}_C$ will not transfer $e1$ and $\{e3\}$ $\xrightarrow{e1}_C$ will not transfer $e2$.

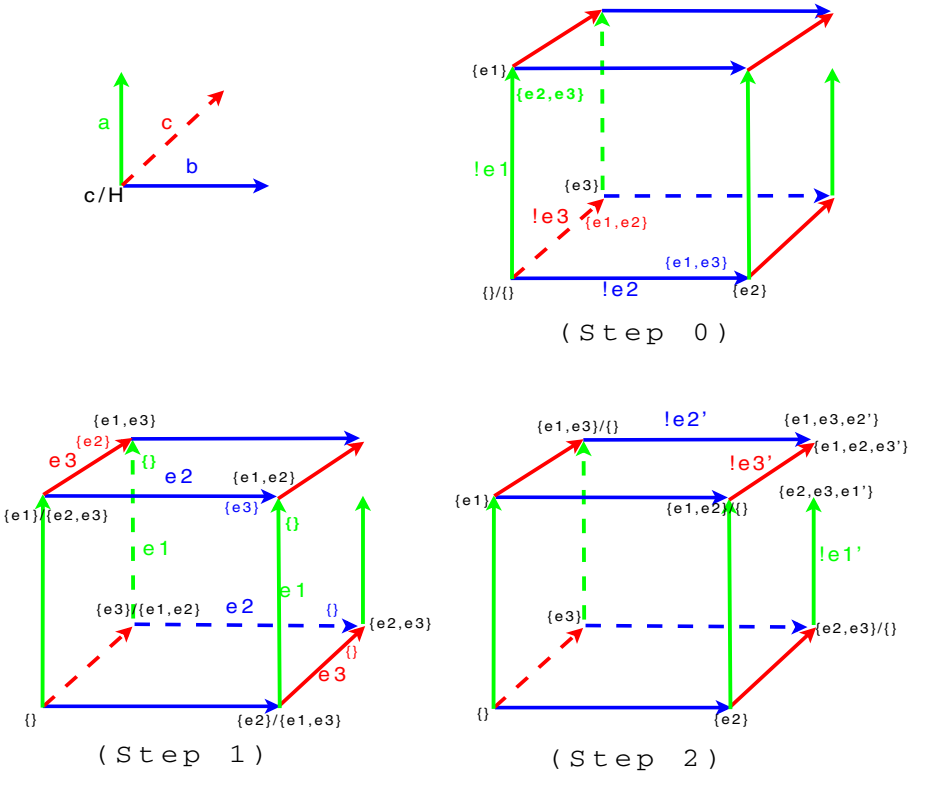


Fig. 9. Unfolding of a broken cube

In the step 2, the set of inherited events at $\{e1, e3\}$ is empty since $e2$ is enabled at $\{e3\}$ but $e2$ is not transferred by $\{e3\}$ $\xrightarrow{e1}$. This vetoes the inheritance of $e2$ at $\{e1, e3\}$, even though $e2$ does get transferred on $\{e1\}$ $\xrightarrow{e3}$. Thus we have to create a second event for action b at $\{e1, e3\}$, i.e. $e2'$. Similarly $e1'$ and $e3'$ are created

for a and c again. So the result is a CS with three maximal configurations. Two of them, $\{e1, e3, e2'\}$ and $\{e1, e2, e3'\}$, are mapped to a same deadlock state.

Note that the procedure is non-terminating for cyclic graphs, in which case the output should be an infinite event system.

Proposition 3. *$\mathcal{U}(L) = (E, C, lb, st)$ is an unfolding of L.*

Proof. Use induction on the size of c' we can prove that st is a function and condition c is true. The induction step needed is that given $c_1 \xrightarrow{e_1}_C c'$ and $c_2 \xrightarrow{e_2}_C c'$, $st(c_1) \xrightarrow{lb(e_1)}_\Delta s_1$ implies $st(c_2) \xrightarrow{lb(e_2)}_\Delta s_2$ and $s_1 = s_2$. To prove the induction step, we need commutativity argument as well as the fact that there exists $c \in C$ such that $c \subseteq c_1 \cap c_2$, $e_1, e_2 \in eb(c)$ and $lb(e_1)||_{st(c)}lb(e_1)$. The fact can be proved because $cs_0 = (E, C) \restriction_{c'}$ is a conflict-free CS and $e1$ cannot block $e2$.

The above also leads to the fact that any pair of configurations in C with subsethood relation must be connected by a path. Based on the fact and using some commutativity argument we can prove closure under bounded union.

Condition b is preserved due to line 6-8 and line 14-16 in the code, while condition d is preserved due to line 11-13 in the code.

Discussion: A major difference of our work from [19,10,14] lies in that our procedure dynamically assigns the independence relations to transitions. In the left graph of Figure 10, transitions $s1 \xrightarrow{c} s4$ and $s1 \xrightarrow{b} s5$ are assigned to be independent if $s1$ is reached from $s0$ via d (since the b, c and d transitions form a cube); otherwise (i.e. $s1$ is reached from $s0$ via a) they are assigned to be dependent.[10] The right graph of Figure 10 gives the unfolding of the left graph. Let it be cs. It is obvious that $s1 \xrightarrow{c} s4$ and $s1 \xrightarrow{b} s5$ are assigned to be $e3$ and $e2$ resp. but $\{e2, e3\}$ is consistent in $cs/\{e4\}$ ($s1$ is reached via d) and inconsistent in $cs/\{e1\}$ ($s1$ is reached via a).

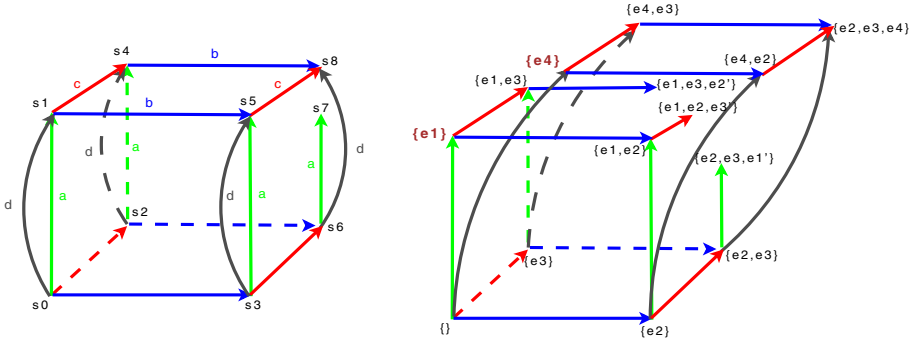


Fig. 10. Dynamic independence on transitions

[10] Note that this is exactly the situation in Figure 9.

5 Correspondence

Based on the unfolding of L into lcs, we can establish the correspondence between extended knots of lcs and stubborn sets of L.

Given $lcs = (E, C, lb, st)$, we say lcs is free of *auto-concurrency* if, for any pair of events $e, e' \in E$ with $lb(e) = lb(e')$, there is no $c \in C$ such that $c \xrightarrow{\{e,e'\}}_C$. We say lcs is free of *auto-conflict* if, for all $c \in C$, there is no ICS K at c so that K contains a pair of events with the same label.

Proposition 4. *$lcs = (E, C, lb, st)$ is an unfolding of L implies lcs is free of auto-concurrency and auto-conflict.*

Proof. It is obvious since e and e' are contained in an ICS K at c implies both are enabled at c. Due to requirement b, i.e. bijection, in the unfolding definition, no two elements of K can have the same label. Similarly we can prove freedom of auto-concurrency.

Theorem 2. *Given any $L = (S, \Sigma, \Delta, \hat{s})$ and any unfolding $lcs = (E, C, lb, st)$ of L, $T \subseteq \Sigma$ is a persistent set at $\hat{s}$ iff there is a knot $K_0 \subseteq E$ such that $lb(K_0) = T$.*

Proof. From Knots to PS (Persistent Set): Let K_0 be a knot of lcs, $T = lb(K_0)$ and $e \in eb(\{\}) \setminus K_0$ be an initial event outside the knot. Then at state $st(\{\}) = \hat{s}$, $lb(e)$ and $lb(e')$ are conditional independent for all $e' \in K_0$ (since $\{e, e'\}$ is consistent due to K_0 being a knot). Since lb is a bijection between $eb(\{\}) \setminus K_0$ and $eb(\hat{s}) \setminus T$ the configurations reachable by executing an event from $eb(\{\}) \setminus K_0$ covers all the states reachable by executing an action from $eb(\hat{s}) \setminus T$. For instance, say $\{e\}$ with $e \in eb(\{\}) \setminus K_0$ covers s with $\hat{s} \xrightarrow{lb(e)} s$. Obviously $K_0 \subseteq eb(\{e\})$. From s on every state reachable by executing one step of $eb(s) \setminus T$ transition is covered by some configuration $\{e, e'\}$ with $e' \in eb(\{e\}) \setminus K_0$ due to the bijection. By induction we can say every state s' reachable from $\hat{s}$ by executing $\Sigma \setminus T$ transitions is covered by a configuration c with $c \cap K_0 = \{\}$ and on all such s' any pair of enabled inside (w.r.t. T) and enabled outside transitions are conditionally independent. This combined with the fact that all T transitions are enabled on such s' implies T is a PS.

From PS to Knots: Let T ($\subseteq eb(\hat{s})$) be a PS at $\hat{s}$ of L and K_0 be the set of events mapped to T by lb from $eb(\{\})$ of lcs. Let us assume K_0 is not a knot. Then there is an event $e \in K_0$ and a configuration $c \in C$ with $c \cap K_0 = \{\}$ such that $c \cup \{e\}$ is inconsistent. Then on the path from $\{\}$ to c, we can find $e' \in c$ and a configuration $c_0 \subseteq c$ such that $c_0 \xrightarrow{e'}$ disables e. Due to freedom of auto-concurrency and auto-conflict, all the events executed on the path are events labelled by actions from outside T. Thus there is a corresponding path in L from $\hat{s}$ to $st(c)$ which executes only actions from outside T. At $st(c)$, $lb(e)$ is disabled, which contradict T being a PS.

Corollary 2. *Given any $L = (S, \Sigma, \Delta, \hat{s})$ and any unfolding $lcs = (E, C, lb, st)$ of L, $T \subseteq \Sigma$ is a stubborn set at $\hat{s}$ implies there is an extended knot $K_1 \subseteq E$ such that $lb(K_1) = T$.*

Proof. Let T be a SS at $\hat{s}$ and K_0 be the set of events labelled by $T \cap eb(\hat{s})$ at $\{\}$. We only need to prove that, for all $a \in T \setminus eb(\hat{s})$, if $a = lb(e)$, then K_0 can block e. It is obvious since otherwise a can be enabled without taking any transition labelled by $lb(K_0)$.

Note that the implication above holds only in one direction: some extended knot may not correspond to any stubborn set. However, thanks to the proposition below, this observation does not change the fact that *the set of actions produced by a residue are indeed those produced by an extended knot that corresponds to a stubborn set.* A branching cell is therefore a long-lived stubborn set.

Proposition 5. *Given any unfolding $lcs = (cs, lb, st)$ of L and any branching cell B of cs, we have, for all $a \in lb(B) \setminus lb(B \cap eb(\{\}))$ and $e \in E$, $lb(e) = a$ implies $B \cap eb(\{\})$ can block e.*

Proof. Use Lemma 6 and the freedom of auto-concurrency.

6 Conclusion

In this paper we have shown that the theory of partial order reduction developed for interleaving setting can be lifted into the world of non-interleaving. A stubborn set (or persistent set) corresponds to an extended knot (or knot), a new notion we developed using well-formed configuration structures. Extended knots are closely related to the existing notion of branching cells from non-interleaving literatures. We prove that a branching cell is actually a 'long-lived' extended knot. Thus we establish an important link between non-interleaving semantic theory and partial order reduction, making possible cross-fertilisation for future works.

Acknowledgements. We thank anonymous referees whose detailed comments greatly improve the presentation of the paper.

References

1. Abbes, S., Benveniste, A.: True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. Information and Computation 204(2), 231–274 (2006)
2. Clarke, E.M., Grumberg, O., Minea, M., Peled, D.: State space reduction using partial order techniques. STTT 2(3), 279–287 (1999)
3. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems. LNCS, vol. 1032. Springer, Heidelberg (1996)
4. Groote, J.F., Sellink, M.P.A.: Confluence for process verification. Theoretical computer science 170(1-2), 47–81 (1996)
5. Gunawardena, J.: Causal automata. TCS 101(2), 265–288 (1992)
6. Liu, X., Walker, D.: Partial confluence of proceses and systems of objects. TCS 206(1-2), 127–162 (1998)

7. Mazurkiewicz, A.: Trace theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 279–324. Springer, Heidelberg (1987)
8. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. In: Kahn, G. (ed.) Semantics of Concurrent Computation. LNCS, vol. 70, pp. 266–284. Springer, Heidelberg (1979)
9. Peled, D.A.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993)
10. Sassone, V., Nielsen, M., Winskel, G.: Models for concurrency: Towards a classification. Theor. Comput. Sci. 170(1-2), 297–348 (1996)
11. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) APN 1990. LNCS, vol. 483, pp. 491–515. Springer, Heidelberg (1991)
12. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998)
13. Valmari, A.: Stubborn set methods for process algebras. In: POMIV 1996. DIMACS Series in DM&TCS, vol. 29, pp. 213–231 (1997)
14. van Glabbeek, R.J.: History preserving process graphs (1996), `http://kilby.stanford.edu/~rvg/pub/history.draft.dvi`
15. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures, event structures and petri nets. Theoretical Computer Science 410(41), 4111–4159 (2009)
16. van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica 37(4), 229–327 (2001)
17. Varacca, D., Völzer, H., Winskel, G.: Probabilistic event structures and domains. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 481–496. Springer, Heidelberg (2004)
18. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)
19. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of logic in Computer Science, vol. 4. Clarendon Press, Oxford (1995)
20. Yakovlev, A., Kishinevsky, M., Kondratyev, A., Lavagno, L., Pietkiewicz-Koutny, M.: On the models for asynchronous circuit behaviour with or causality. FMSD 9(3), 189–233 (1996)

On Parametric Steady State Analysis of a Generalized Stochastic Petri Net with a Fork-Join Subnet*

Jonathan Billington and Guy Edward Gallasch

Distributed Systems Laboratory
University of South Australia
Mawson Lakes Campus, SA, 5095, Australia
{jonathan.billington,guy.gallasch}@unisa.edu.au

Abstract. The performance analysis of parallel systems which are synchronised is an important but challenging task. This is because product form solutions are not available when synchronisation is required. The system of interest is a controlled fork-join network, comprising two parallel processes. Each process is governed by an exponentially distributed delay. The system has a capacity, N, which cannot be exceeded. Thus only N requests for service can be handled concurrently, so that further requests are blocked. The arrival of requests is also governed by an exponential distribution. We model this class of system with a Generalized Stochastic Petri Net (GSPN) which includes a two branch fork-join structure controlled by an environment that enforces the capacity constraint. This GSPN is thus parameterised by the capacity, N. We derive the parametric reduced reachability graph for arbitrary N, and show that it has $(N+1)^2$ markings and one strongly connected component. We also prove that the GSPN is bounded by N, and thus show that one process cannot lead the other by more than N. We obtain the associated family of continuous time Markov chains, and derive the family of global balance equations for arbitrary N. We solve these equations for the steady state probabilities for $N = 1$ and 2, and then present a theorem for the general form of the solution for arbitrary $N > 2$ in terms of ratios of polynomials in the transition rates. A scheme of 21 relationships between these polynomials is also obtained. Finally we explore some asymptotic behaviour of the steady state probabilities and relate them to previously obtained closed form approximate solutions.

Keywords: Parametric Performance Analysis, Generalized Stochastic Petri Nets, Parallel Processes, Markov Chains, Asymptotic Results.

1 Introduction

Generalized Stochastic Petri Nets (GSPNs) [1, 3] are very useful for modelling parallel systems, which are important in many domains including manufacturing

* Supported by Australian Research Council Discovery Project Grant DP0880928.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 268–287, 2011.

and computing. The *fork-join* construct [2] is used when the input to and output from parallel processes needs synchronisation. Unfortunately, the performance analysis of systems with synchronisation is difficult as product form solutions are generally not available. Due to their importance, fork-join systems have been extensively studied [2, 6, 7, 11, 12, 14, 15, 16, 19, 20, 21, 22]. However, we are not aware of work that considers the *parametric* performance analysis of GSPNs which include a fork-join subnet. The GSPN of interest includes a *fork-join* subnet that represents two parallel processes that need to be synchronised, and an environment which enforces a capacity constraint, N, on each process ensuring that one process can never be ahead of the other by more than N items. The environment also models requests for service from the two parallel processes, that are blocked if there are more requests than the contraint will allow.

Our overall goal is to derive performance measures for this system, such as the mean amount one process is ahead of another or the mean input or output buffer size for each process. To do this we can use steady state Markov analysis [1, 3]. However, we also want to know how this system behaves as its capacity, N, is varied. Thus N is considered as a positive integer parameter of the system. The article closest to ours [13] considers aggregating the fork-join subnet and hence cannot calculate any performance measures within the fork-join. In recent work [4, 9] we presented significant numerical studies of our GSPN to demonstrate and characterise the convergence of the steady state probabilities as N grows. [9] presented a heuristic to estimate the value of N by which convergence had occurred to a certain number of decimal places, while in [4] we used observed geometric progressions in the probabilities to derive closed form approximate solutions when the fork-join is fully loaded (i.e. the rate of requests is greater than the minimum rate of the processes).

This paper investigates the *parametric analysis* of our GSPN model with the aim of obtaining *exact analytical* solutions for its steady state probabilities for arbitrary capacity N and to provide some theoretical underpinning of the results obtained in [4]. The GSPN comprises two parallel branches where each branch includes a single exponential transition with a fixed but arbitrary firing rate. We provide the general theory for the parametric analysis of this GSPN. In particular, expressions are derived for the family of reduced reachability graphs[1] which satisfy various properties. We then obtain the corresponding parametric continuous time Markov chain (CTMC) which allows us to formulate the general set of global balance equations for the steady state probabilities of the system for arbitrary N. Solutions are presented for $N = 1$ and 2 in terms of the symbolic firing rates. We then provide the general form of the solution for arbitrary $N \geq 3$. We illustrate the general solution for the case $N = 3$, and derive some asymptotic results for arbitrary N.

Several contributions are provided. Firstly, we state and prove a theorem that defines the parametric reduced reachability graph (RRG_N) of the GSPN. This is a key result for parametric analysis. Secondly, we prove that the GSPN is

[1] A reduced reachability graph is one that only has tangible markings[1, 8, 13], sometimes called a Tangible Reachability Graph or RRG-0[1].

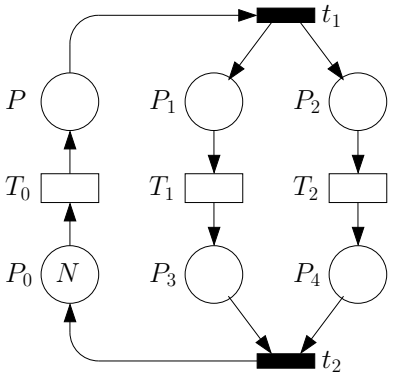


Fig. 1. GSPN with a Fork-Join Subnet

bounded by N and show that each process satisfies its capacity constraint and that no process can be more than N items ahead of the other. Thirdly, two properties of RRG_N are proved: the number of states is given by $(N+1)^2$, and it comprises one strongly connected component. Fourthly, we define an integer encoding of the markings of RRG_N. The encoding provides a convenient numbering of the states for the corresponding family of CTMCs, which we prove are irreducible. Fifthly, from the parametric CTMC we derive a set of 11 equations that represent the family of global balance equations for arbitrary N and present solutions for the steady state probabilities for $N = 1, 2$. Sixthly, we state and prove the main result of the paper: a theorem that defines the general form of the solution for the steady state probabilities in terms of ratios of polynomials in the firing rates. This allows us to obtain a set of 21 simple relationships between these polynomials. Finally we derive some asymptotic properties of the steady state probabilities from the general form of the solution and relate them to the approximate closed form approximations obtained in [4].

The rest of the paper is organised as follows. Section 2 describes the GSPN of interest. The parametric reduced reachability graph and its properties are derived in Section 3. Section 4 presents several results using Markov analysis. Conclusions are drawn in Section 5. We assume some familiarity with GSPNs and their analysis [1, 3].

2 GSPN Model

We consider the GSPN shown in Fig. 1. This GSPN includes a fork-join subnet with 2 branches (P_1, T_1, P_3 and P_2, T_2, P_4) between immediate transitions, t_1 (the fork) and t_2 (the join), represented by bars. The remaining transitions, represented by rectangles, are exponentially distributed timed transitions with their own rates, i.e. λ_i is associated with T_i, $i \in \{0, 1, 2\}$. T_0 represents customer requests. Initially all places are empty except for P_0 which contains N tokens, limiting the capacity of the fork-join to N customer requests. A customer request

results in a request for processing to each of the parallel processes being deposited in their buffers, P_1 and P_2, via the fork. To free up capacity, as soon as two different items (one created by each process) are available, they are removed immediately (transition t_2) for further processing.

3 Parametric Reduced Reachability Graph

We denote the Parametric RRG of the GSPN of Fig. 1 by $RRG_N = (TS_N, A_N)$ where TS_N is the set of reachable tangible markings, and $A_N \subseteq TS_N \times T \times TS_N$ is the set of arcs between tangible markings, with T the set of timed transitions. Each marking in TS_N can be represented by a vector, $(M(P_0), M(P_1), M(P_2), M(P_3), M(P_4))$. Because $M_0(P) = 0$, t_1 ensures that $\forall M \in TS_N, M(P) = 0$, hence we exclude $M(P)$ from our marking vector.

Theorem 1. *$RRG_N = (TS_N, A_N)$, where*
$TS_N = \{(N,0,0,0,0), (N-x,x,x,0,0), (N-x,x-y,x,y,0), (N-x,x,x-y,0,y) \mid 1 \leq x \leq N, 1 \leq y \leq x\}$ and A_N is given by Table 1.

Table 1. The arcs, A_N, of RRG_N

Source	Label	Destination	Conditions
$(N-x,x,x,0,0)$	T_0	$(N-x-1,x+1,x+1,0,0)$	$0 \leq x < N$
$(N-x,x-y,x,y,0)$	T_0	$(N-x-1,x-y+1,x+1,y,0)$	$1 \leq x < N, 1 \leq y \leq x$
$(N-x,x,x-y,0,y)$	T_0	$(N-x-1,x+1,x-y+1,0,y)$	
$(N-x,x-y,x,y,0)$	T_1	$(N-x,x-y-1,x,y+1,0)$	$1 \leq x \leq N, 0 \leq y < x$
$(N-x,x,x-y,0,y)$	T_2	$(N-x,x,x-y-1,0,y+1)$	
$(N-x,x-y,x,y,0)$	T_2	$(N-x+1,x-y,x-1,y-1,0)$	$1 \leq x \leq N, 1 \leq y \leq x$
$(N-x,x,x-y,0,y)$	T_1	$(N-x+1,x-1,x-y,0,y-1)$	

Proof. The proof comprises two lemmas. Firstly, we prove that all tangible markings in TS_N are reachable from the initial marking. Then we prove that no other tangible marking is reachable from the markings in TS_N by firing every enabled transition in all markings in TS_N. This also shows that the arcs of the Parametric RRG are given by A_N.

Lemma 1. (SPANNING) *All tangible markings in TS_N are reachable from M_0, i.e. $TS_N \subseteq [M_0\rangle$.*

Proof. From the initial marking, $M_0 = (N,0,0,0,0)$, T_0 and then t_1 are fired N times, until $M(P_0) = 0$, resulting in the set of tangible markings:

$$Depth_N = \{(N-x,x,x,0,0) \mid 0 \leq x \leq N\} \tag{1}$$

These markings occur along the horizontal centre line of RRG_N (see $CTMC_N$ in Fig. 2), starting from the initial marking and moving deeper (to the right)

into RRG_N. From each marking in $Depth_N$ (with the exception of the initial marking), T_1 can be fired repeatedly until $M(P_1) = 0$. From Equation (1), the new markings obtained are denoted by $Offset_N^+$ (markings above the horizontal centre line, hence with a 'positive offset') and given by:

$$Offset_N^+ = \{(N-x, x-y, x, y, 0) \mid 1 \leq x \leq N, 1 \leq y \leq x\} \tag{2}$$

All of these markings are tangible as neither t_1 nor t_2 are enabled. Similarly, from each marking in $Depth_N$ (except M_0), T_2 can be fired repeatedly until $M(P_2) = 0$, giving the new tangible markings:

$$Offset_N^- = \{(N-x, x, x-y, 0, y) \mid 1 \leq x \leq N, 1 \leq y \leq x\} \tag{3}$$

The union of these three sets is $Depth_N \cup Offset_N^+ \cup Offset_N^- = TS_N$ □

Corollary 1. *$Depth_N$, $Offset_N^+$ and $Offset_N^-$ are mutually exclusive.*

Lemma 2 (Containment). *TS_N contains all tangible markings reachable from M_0, i.e. $[M_0\rangle \subseteq TS_N$, and A_N comprises all arcs of RRG_N.*

Proof. Firstly, observe that: T_0 is enabled by any tangible marking in which $M(P_0) > 0$; T_1 is enabled when $M(P_1) > 0$; and T_2 when $M(P_2) > 0$.

Arcs Labelled by T_0:
The subset of TS_N that enables T_0 is given by:

$$\begin{aligned} EN_{(T_0,N)} = &\{(N-x, x, x, 0, 0) \mid 0 \leq x < N\} \\ &\cup \{(N-x, x-y, x, y, 0), (N-x, x, x-y, 0, y) \\ &\quad \mid 1 \leq x < N, 1 \leq y \leq x\} \end{aligned}$$

The set of arcs corresponding to the occurrence of T_0 in RRG_N is given by firing T_0 and then t_1 in each marking in $EN_{(T_0,N)}$:

$$\begin{aligned} Arcs_{(T_0,N)} = &\{((N-x, x, x, 0, 0), T_0\ t_1, (N-x-1, x+1, x+1, 0, 0)) \mid 0 \leq x < N\} \\ &\cup \{((N-x, x-y, x, y, 0), T_0\ t_1, (N-x-1, x-y+1, x+1, y, 0)), \\ &\quad ((N-x, x, x-y, 0, y), T_0\ t_1, (N-x-1, x+1, x-y+1, 0, y)) \\ &\quad \mid 1 \leq x < N, 1 \leq y \leq x\} \end{aligned}$$

which corresponds to the first three rows of Table 1 after dropping t_1 from the arc labels (as immediate transitions are not part of an RRG). The destination markings are given by:

$$\begin{aligned} Dest(Arcs_{(T_0,N)}) &= \{(N-x-1, x+1, x+1, 0, 0) \mid 0 \leq x < N\} \\ &\quad \cup \{(N-x-1, x-y+1, x+1, y, 0), \\ &\qquad (N-x-1, x+1, x-y+1, 0, y) \mid 1 \leq x < N, 1 \leq y \leq x\} \\ &= \{(N-x, x, x, 0, 0) \mid 1 \leq x \leq N\} \\ &\quad \cup \{(N-x, x-y, x, y, 0), (N-x, x, x-y, 0, y) \\ &\qquad \mid 1 < x \leq N, 1 \leq y < x\} \\ &\subset TS_N \end{aligned}$$

Arcs Labelled by T_1:
The subset of TS_N which enables T_1 is given by:

$$\begin{aligned} EN_{(T_1,N)} &= \{(N-x, x-y, x, y, 0) \mid 1 \le x \le N, 0 \le y < x\} \\ &\cup \{(N-x, x, x-y, 0, y) \mid 1 \le x \le N, 1 \le y \le x\} \end{aligned}$$

where we have combined the specification of the subsets which enable T_1 in both $Depth_N$ and $Offset_N^+$ in the first set. The set of arcs corresponding to the occurrence of T_1 in RRG_N is given by firing T_1 from each marking in $EN_{(T_1,N)}$, followed by t_2 when $M(P_4) > 0$:

$$\begin{aligned} Arcs_{(T_1,N)} = \{&((N-x, x-y, x, y, 0), T_1, (N-x, x-y-1, x, y+1, 0)) \\ &\mid 1 \le x \le N, 0 \le y < x\} \\ \cup \{&((N-x, x, x-y, 0, y), T_1\ t_2, (N-x+1, x-1, x-y, 0, y-1) \\ &\mid 1 \le x \le N, 0 \le y \le x\} \end{aligned}$$

which corresponds to rows 4 and 7 of Table 1 after dropping t_2 from the relevant arc labels. The destination markings are given by:

$$\begin{aligned} Dest(Arcs_{(T_1,N)}) &= \{(N-x, x-y-1, x, y+1, 0) \mid 1 \le x \le N, 0 \le y < x\} \\ &\cup \{(N-x+1, x-1, x-y, 0, y-1) \mid 1 \le x \le N, 1 \le y \le x\} \\ &= \{(N-x, x-y, x, y, 0) \mid 1 \le x \le N, 1 \le y \le x\} \\ &\cup \{(N-x, x, x-y, 0, y) \mid 0 \le x < N, 0 \le y \le x\} \\ &\subset TS_N \end{aligned}$$

Arcs Labelled by T_2:
The subset of TS_N which enables T_2 is given by:

$$\begin{aligned} EN_{(T_2,N)} &= \{(N-x, x-y, x, y, 0) \mid 1 \le x \le N, 1 \le y \le x\} \\ &\cup \{(N-x, x, x-y, 0, y) \mid 1 \le x \le N, 0 \le y < x\} \end{aligned}$$

The set of arcs corresponding to the occurrence of T_2 in RRG_N is given by firing T_2 from each marking in $EN_{(T_2,N)}$, followed by t_2 when $M(P3) > 0$:

$$\begin{aligned} Arcs_{(T_2,N)} = \{&((N-x, x, x-y, 0, y), T_2, (N-x, x, x-y-1, 0, y+1) \\ &\mid 1 \le x \le N, 0 \le y < x\} \\ \cup \{&((N-x, x-y, x, y, 0), T_2\ t_2, (N-x+1, x-y, x-1, y-1, 0)) \\ &\mid 1 \le x \le N, 1 \le y \le x\} \end{aligned}$$

which corresponds to rows 5 and 6 of Table 1 after dropping t_2. The destination markings are given by:

$$\begin{aligned} Dest(Arcs_{(T_2,N)}) &= \{(N-x, x, x-y-1, 0, y+1) \mid 1 \le x \le N, 0 \le y < x\} \\ &\cup \{(N-x+1, x-y, x-1, y-1, 0) \mid 1 \le x \le N, 1 \le y \le x\} \\ &= \{(N-x, x, x-y, 0, y) \mid 1 \le x \le N, 1 \le y \le x\} \\ &\cup \{(N-x, x-y, x, y, 0) \mid 0 \le x < N, 0 \le y \le x\} \\ &\subset TS_N \end{aligned}$$

Hence, the lemma is proved. □

Lemma 1 states that all tangible markings given in TS_N are reachable from the initial marking and Lemma 2 that no other tangible markings can be reached. Thus, TS_N is exactly the set of reachable tangible markings of the GSPN in Fig. 1. Lemma 2 also fires every enabled timed transition from every tangible marking in TS_N, giving rise to every arc in A_N. Thus the theorem is proved. □

Lemma 3. *RRG_N contains $(N+1)^2$ markings: $|TS_N| = (N+1)^2$.*

Proof. From Corollary 1, TS_N can be partitioned into the 3 sets, $Depth_N$, $Offset_N^+$, and $Offset_N^-$, the sizes of which are given by:

- $|\{(N-x, x, x, 0, 0) \mid 0 \leq x \leq N\}| = N+1$;
- $|\{(N-x, x-y, x, y, 0) \mid 1 \leq x \leq N, 1 \leq y \leq x\}| = \sum_{x=1}^{N} x = N(N+1)/2$; and similarly
- $|\{(N-x, x, x-y, 0, y) \mid 1 \leq x \leq N, 1 \leq y \leq x\}| = \sum_{x=1}^{N} x = N(N+1)/2$.

Adding these together gives the desired result: $|TS_N| = (N+1)^2$ □

Lemma 4. *(Place Bounds) $\forall M \in TS_N, \forall p \in \{P_0, P_1, P_2, P_3, P_4\}, M(p) \leq N$.*

Proof. From Theorem 1, substituting for all allowed values of x and y in TS_N reveals that $\forall M \in TS_N, \forall p \in \{P_0, P_1, P_2, P_3, P_4\}, M(p) \leq N$. □

Because P is bounded by 1, due to the immediate transition t_1, then it follows from lemma 4 that the GSPN is bounded by N. We could also derive that the GSPN is bounded by N from the fact that the underlying Place/Transition net is a Marked Graph [17] or T-system [5], by observing that it comprises two circuits passing through P_0, resulting in the following place invariants:
$\forall M \in [M_0\rangle,\ M(P_0) + M(P) + M(P_1) + M(P_3) = N$ and
$\forall M \in [M_0\rangle,\ M(P_0) + M(P) + M(P_2) + M(P_4) = N$.

The reason for including the stronger result of Lemma 4 is that we need the specific bounds on each of the places P_1 to P_4 (see below). Note that it is very easy to prove directly from Theorem 1[2].

Lemma 5. $\forall M \in TS_N,\ M(P_1) + M(P_3) \leq N$.

Lemma 6. $\forall M \in TS_N,\ M(P_2) + M(P_4) \leq N$.

Proof. Lemmas 5 and 6 follow directly from the above place invariants. □

Lemmas 5 and 6 ensure that the capacity constraint of N on each process is met and that either process can be ahead of the other by at most N. Lemma 4 implies that in the worst case, one process can be ahead of the other by N.

[2] Further, the main point of obtaining Theorem 1 is not to prove boundedness nor the size of the RRG, but to be able to obtain the parametric Markov chain and its global balance equations in section 4. However, it is useful for all these purposes.

Lemma 7. *RRG_N has one strongly connected component.*

Proof. Having a single strongly connected component is equivalent to every marking in TS_N being a home marking. Because the underlying Place/Transition net of the GSPN is a T-system, we can use a result from [5] (Proposition 8.2) that every reachable marking of a live T-system is a home marking. Firstly we note that the T-system is live, because the two circuits $P_0T_0Pt_1P_1T_1P_3t_2$ and $P_0T_0Pt_1P_2T_2P_4t_2$ are initially marked as $M_0(P_0) = N$ (see Theorem 3.15 of [5]). Thus every marking of the GSPN's T-system is a home marking. Removing vanishing markings does not affect this result, so every marking in TS_N is a home marking and hence RRG_N has one strongly connected component. □

4 Parametric Markov Analysis

4.1 Family of Markov Chains

We can convert RRG_N into the corresponding CTMC state transition rate diagram, $CTMC_N$.

Theorem 2. *$CTMC_N = (S_N, E_N)$, where $S_N = \{1, 2, ..., (N+1)^2\}$ is the set of states of $CTMC_N$ and E_N is its set of directed edges given by Table 1, where $\forall i \in \{0, 1, 2\}, T_i$ is replaced by λ_i, and the source and destination markings are transformed into the corresponding source and destination states by:*

$$ID_N \ : \ TS_N \rightarrow \{1, 2, \ldots, (N+1)^2\}$$

where

$$ID_N(M) = (N - M(P_0))^2 + (N - M(P_0)) - M(P_3) + M(P_4) + 1$$

Proof. Follows from Lemma 3 and that for all N, $CTMC_N$ is isomorphic to RRG_N [1, 3]. □

The bijection on states always maps the initial marking, $M_0 = (N, 0, 0, 0, 0)$, to state 1 for every N. Figure 2 depicts $CTMC_N$. The above bijection numbers the states sequentially from 1 to $(N+1)^2$ in successive vertical columns as we proceed left to right and top to bottom from the initial state. This makes it easier to refer to states and is useful in tools such as Matlab.

Lemma 8. *Each member of the family, $CTMC_N$, is irreducible.*

Proof. Follows immediately from Lemma 7 and Theorem 2. □

4.2 Global Balance Equations

From $CTMC_N$ in Fig. 2 we can now derive the set of *global balance equations* (GBEs) [1]. The set of GBEs comprises $(N+1)^2$ equations, one for each state.

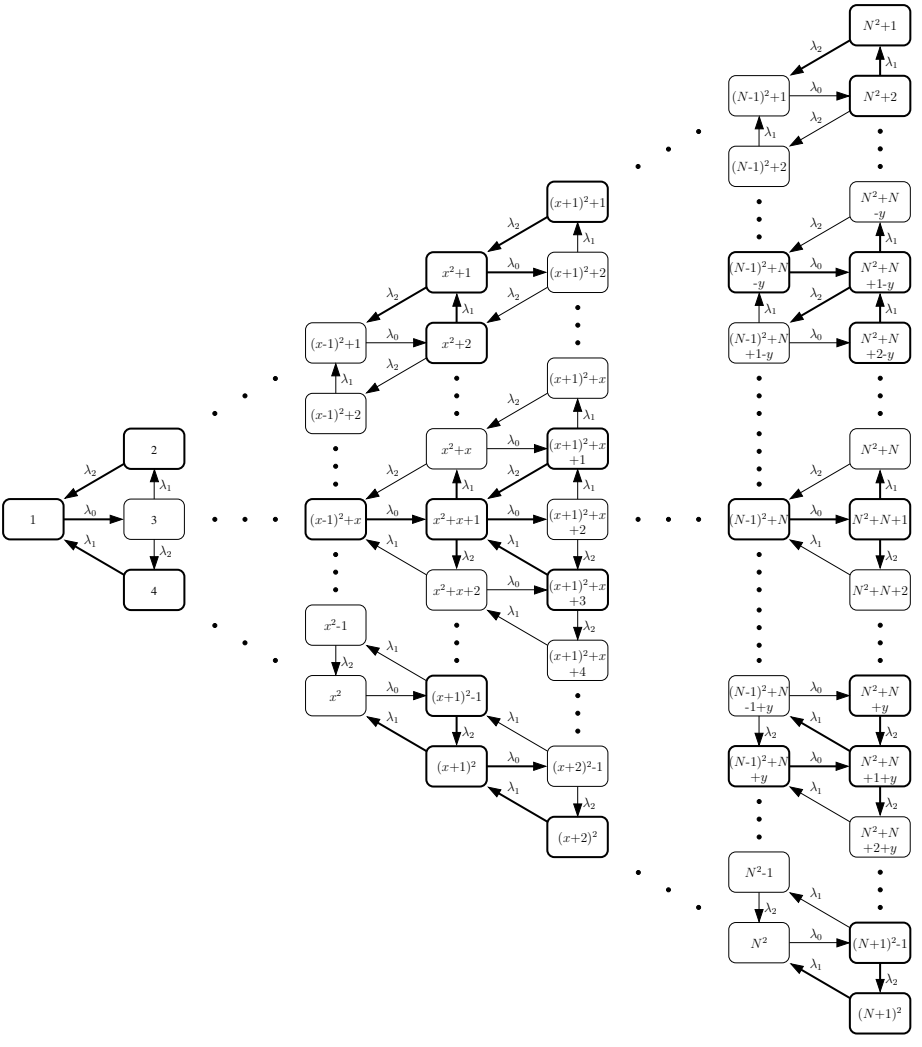


Fig. 2. The Parametric Continuous Time Markov Chain, $CTMC_N$, for the closed fork-join GSPN in Fig. 1

However, we can capture all the GBEs with just 11 equations. This is illustrated in Fig. 2 by considering the bold states and arcs. In the following we denote the steady state probability of being in state i in $CTMC_N$ by $\pi_{(N,i)}$. The first 4 equations given below are the only ones required when $N = 1$.

$$\lambda_0\pi_{(N,1)} = \lambda_1\pi_{(N,4)} + \lambda_2\pi_{(N,2)} \tag{4}$$

$$\lambda_2\pi_{(N,N^2+1)} = \lambda_1\pi_{(N,N^2+2)} \tag{5}$$

$$(\lambda_1 + \lambda_2)\pi_{(N,N^2+N+1)} = \lambda_0\pi_{(N,N^2-N+1)} \tag{6}$$

$$\lambda_1\pi_{(N,(N+1)^2)} = \lambda_2\pi_{(N,(N+1)^2-1)} \tag{7}$$

Equation (4) is the equation for state 1. Equation (5), is for the top right state ((N^2+1)), Equation (6), relates to the right-most state on the horizontal centre row ((N^2+N+1)) and Equation (7) is for the bottom right state ($(N+1)^2$). Equations (8) and (9) complete the GBEs for the far right column of states.

$$(\lambda_1+\lambda_2)\pi_{(N,N^2+N+1-y)} = \lambda_0\pi_{(N,N^2-N+1-y)} + \lambda_1\pi_{(N,N^2+N+2-y)},\ 0<y<N \tag{8}$$

$$(\lambda_1+\lambda_2)\pi_{(N,N^2+N+1+y)} = \lambda_0\pi_{(N,N^2-N+1+y)} + \lambda_2\pi_{(N,N^2+N+y)},\ 0<y<N \tag{9}$$

For $N=1$ there are no values that y can take, and so these equations do not exist. For $N=2$, we just get two equations (for states 6 and 8), as y can only take the value 1.

Equation (10) defines the GBEs for the states along the top boundary of the triangle, minus the end points, already captured by Equations (4) and (5).

$$(\lambda_0+\lambda_2)\pi_{(N,x^2+1)} = \lambda_1\pi_{(N,x^2+2)} + \lambda_2\pi_{(N,(x+1)^2+1)}, 0<x<N \tag{10}$$

$$\begin{aligned}(\lambda_0+\lambda_1+\lambda_2)\pi_{(N,x^2+x+1-y)} = {} & \lambda_0\pi_{(N,x^2-x+1-y)} + \lambda_1\pi_{(N,x^2+x+2-y)} \\ & + \lambda_2\pi_{(N,(x+1)^2+x+1-y)}, 0<x<N, 0<y<x\end{aligned} \tag{11}$$

Equation (11) (not illustrated in Fig. 2) captures the GBEs for all non-boundary states in the upper triangle excluding the horizontal centre row of states. For $N=1$ and 2, x and/or y cannot take a value, and hence the equation does not exist. Thus these equations only come into play for $N>2$. For $N=3$ we only have one non-boundary state in the upper triangle ($x=2$, $y=1$) however, for larger N, these equations (and those derived from Equation (13) below) form the bulk of the GBEs.

$$\begin{aligned}(\lambda_0+\lambda_1+\lambda_2)\pi_{(N,x^2+x+1)} = {} & \lambda_0\pi_{(N,x^2-x+1)} + \lambda_1\pi_{(N,(x+1)^2+x+3)} \\ & + \lambda_2\pi_{(N,(x+1)^2+x+1)}, 0<x<N\end{aligned} \tag{12}$$

Equation (12) defines the GBEs for the states along the centre row, excluding the end points, already given by Equations (4) and (6).

$$\begin{aligned}(\lambda_0+\lambda_1+\lambda_2)\pi_{(N,x^2+x+1+y)} = {} & \lambda_0\pi_{(N,x^2-x+1+y)} + \lambda_1\pi_{(N,(x+1)^2+x+y+3)} \\ & + \lambda_2\pi_{(N,x^2+x+y)}, 0<x<N, 0<y<x\end{aligned} \tag{13}$$

Equation (13) (also not illustrated in Fig. 2) covers the GBEs for all non-boundary states in the lower triangle, excluding the centre row, analogously with Equation (11).

$$(\lambda_0+\lambda_1)\pi_{(N,(x+1)^2)} = \lambda_1\pi_{(N,(x+2)^2)} + \lambda_2\pi_{(N,(x+1)^2-1)}, 0<x<N \tag{14}$$

Finally, Equation (14) covers the GBEs of the states along the bottom boundary of the triangle, excluding the end points.

4.3 Solving the GBEs

To obtain solutions for the steady-state probabilities of all $(N+1)^2$ states, we need to solve the system of equations above under the condition that the sum of all probabilities must equal 1:

$$\sum_{i=1}^{(N+1)^2} \pi_{(N,i)} = 1 \tag{15}$$

For example, $N = 1$ gives the following set of GBEs:

$$\begin{aligned}
\lambda_0 \pi_{(1,1)} &= \lambda_2 \pi_{(1,2)} + \lambda_1 \pi_{(1,4)} \\
\lambda_2 \pi_{(1,2)} &= \lambda_1 \pi_{(1,3)} \\
\lambda_0 \pi_{(1,1)} &= (\lambda_1 + \lambda_2) \pi_{(1,3)} \\
\lambda_2 \pi_{(1,3)} &= \lambda_1 \pi_{(1,4)}
\end{aligned}$$

Solving the last three equations for $\pi_{(1,1)}$, $\pi_{(1,2)}$ and $\pi_{(1,4)}$ in terms of $\pi_{(1,3)}$ and substituting into Equation (15) gives:

$$\pi_{(1,3)} = \frac{\lambda_0 \lambda_1 \lambda_2}{(\lambda_1 + \lambda_2)(\lambda_1 \lambda_2) + \lambda_0 ({\lambda_1}^2 + \lambda_1 \lambda_2 + {\lambda_2}^2)}$$

from which we then obtain the remaining probabilities:

$$\begin{aligned}
\pi_{(1,1)} &= \frac{(\lambda_1 + \lambda_2) \lambda_1 \lambda_2}{(\lambda_1 + \lambda_2)(\lambda_1 \lambda_2) + \lambda_0 ({\lambda_1}^2 + \lambda_1 \lambda_2 + {\lambda_2}^2)} \\
\pi_{(1,2)} &= \frac{\lambda_0 {\lambda_1}^2}{(\lambda_1 + \lambda_2)(\lambda_1 \lambda_2) + \lambda_0 ({\lambda_1}^2 + \lambda_1 \lambda_2 + {\lambda_2}^2)} \\
\pi_{(1,4)} &= \frac{\lambda_0 {\lambda_2}^2}{(\lambda_1 + \lambda_2)(\lambda_1 \lambda_2) + \lambda_0 ({\lambda_1}^2 + \lambda_1 \lambda_2 + {\lambda_2}^2)}
\end{aligned}$$

We can solve these equations symbolically using Matlab for $N \leq 8$. For example, for $N = 2$: $\pi_{(2,i)} = \frac{C_{(2,i)}}{D_2}$ for $1 \leq i \leq 9$, where

$$\begin{aligned}
C_{(2,1)} &= (\lambda_1 \lambda_2)^2 ((\lambda_1 + \lambda_2)^3 + \lambda_0 ({\lambda_1}^2 + {\lambda_2}^2)) \\
C_{(2,2)} &= \lambda_0 {\lambda_1}^3 \lambda_2 ((\lambda_1 + \lambda_2)^2 + \lambda_0 \lambda_1) \\
C_{(2,3)} &= \lambda_0 (\lambda_1 \lambda_2)^2 (\lambda_1 + \lambda_2)(\lambda_0 + \lambda_1 + \lambda_2) \\
C_{(2,4)} &= \lambda_0 \lambda_1 {\lambda_2}^3 ((\lambda_1 + \lambda_2)^2 + \lambda_0 \lambda_2) \\
C_{(2,5)} &= {\lambda_0}^2 {\lambda_1}^4 (\lambda_0 + \lambda_1 + 2\lambda_2) \\
C_{(2,6)} &= {\lambda_0}^2 {\lambda_1}^3 \lambda_2 (\lambda_0 + \lambda_1 + 2\lambda_2) \\
C_{(2,7)} &= (\lambda_0 \lambda_1 \lambda_2)^2 (\lambda_0 + \lambda_1 + \lambda_2) \\
C_{(2,8)} &= {\lambda_0}^2 \lambda_1 {\lambda_2}^3 (\lambda_0 + 2\lambda_1 + \lambda_2) \\
C_{(2,9)} &= {\lambda_0}^2 {\lambda_2}^4 (\lambda_0 + 2\lambda_1 + \lambda_2)
\end{aligned}$$

and

$$\begin{aligned} D_2 = {} & (\lambda_1\lambda_2)^2(\lambda_1+\lambda_2)^3 \\ & + \lambda_0(\lambda_1\lambda_2)(\lambda_1{}^4 + 4\lambda_1{}^3\lambda_2 + 4\lambda_1{}^2\lambda_2{}^2 + 4\lambda_1\lambda_2{}^3 + \lambda_2{}^4) \\ & + \lambda_0{}^2(\lambda_1{}^5 + 4\lambda_1{}^4\lambda_2 + 4\lambda_1{}^3\lambda_2{}^2 + 4\lambda_1{}^2\lambda_2{}^3 + 4\lambda_1\lambda_2{}^4 + \lambda_2{}^5) \\ & + \lambda_0{}^3(\lambda_1{}^4 + \lambda_1{}^3\lambda_2 + \lambda_1{}^2\lambda_2{}^2 + \lambda_1\lambda_2{}^3 + \lambda_2{}^4) \end{aligned}$$

4.4 General Form of Solution

Above we have presented the steady state probabilities for $N = 1$ and 2. These results are difficult to generalise for arbitrary N, due to complexity of the resulting polynomials in the rates $(\lambda_0, \lambda_1, \lambda_2)$ in the numerator and denominator of each probability (see [10] for $N = 3$ and 4). However, we can discern the general form of the solutions in terms of ratios of polynomials in λ_0, the coefficients of which are polynomials in λ_1 and λ_2.

Theorem 3. *For $N \geq 3$ and $1 \leq i \leq (N+1)^2$, the general form of the solution to the GBEs is given by:*

$$\pi_{(N,i)} = \frac{C_{(N,i)}}{D_N} \tag{16}$$

where

$$C_{(N,i)} = \lambda_0{}^{u(i)}\lambda_1{}^{N+u(i)-v(i)}\lambda_2{}^{N+v(i)-u(i)} \sum_{j=0}^{N^2-N} \lambda_0{}^j c_{(N,i,j)}(N^2-u(i)-j;\lambda_1,\lambda_2) \tag{17}$$

$$D_N = \sum_{i=1}^{(N+1)^2} C_{(N,i)} \tag{18}$$

with

$$u(i) = \mathit{floor}(\sqrt{i-1}) \tag{19}$$

$$v(i) = i - u(i)^2 - 1 \tag{20}$$

and $n = N^2 - u(i) - j$ is the degree of the polynomial $c_{(N,i,j)}(n;\lambda_1,\lambda_2)$, given by

$$\begin{aligned} c_{(N,i,j)}(n;\lambda_1,\lambda_2) = {} & c(N,i,j,n)\lambda_1{}^n + c(N,i,j,n-1)\lambda_1{}^{n-1}\lambda_2 + \cdots \\ & + c(N,i,j,1)\lambda_1\lambda_2{}^{n-1} + c(N,i,j,0)\lambda_2{}^n, \\ & c(N,i,j,n) > 0 \text{ or } c(N,i,j,0) > 0 \end{aligned} \tag{21}$$

Proof (Sketch). Substitute the solution for $\pi_{(N,i)}$ (equation (16)) into each of the 11 GBEs in turn. D_N is a common denominator. Multiply through each GBE by D_N to eliminate it. Substitute for $C_{(N,i)}$ using equation (17) and compute the values of $u(i)$ and $v(i)$ using equations (19) and (20), for the values of i

required. Divide through by common factors of powers of the rates to reveal polynomials of the same degree in each of the rates. Select the coefficients of the polynomials in λ_1 and λ_2 (equation (21)) to ensure that each GBE is satisfied. Finally, substituting for $\pi_{(N,i)}$ (equation (16)) into equation (15) and multiplying through by D_N, shows that equation (18) also holds. □

It is interesting to observe that this general form also holds for $N = 1$, but does not hold for $N = 2$. However, it does hold for $N = 2$ if we replace N^2 by $N^2 - 1$ in equation (17). That is, the degrees of the polynomials are one less than expected, and hence there is one less of them. The reason for this is unclear, but it may be due to $N = 2$ being a special case where only 9 of the 11 GBEs are invoked. This corresponds to all states of the CTMC being either boundary nodes or residing on the centre row. In contrast, for $N \geq 3$, non-boundary states are included that are not on the centre row and all 11 equations are required.

Corollary 2. *The polynomials defined in Theorem 3 have the following relationships:*
For $j \in \{0, 1, \ldots, N^2 - N\}$

1. $c_{(N,1,j)}(N^2-j;\lambda_1,\lambda_2) = \lambda_1 c_{(N,2,j)}(N^2-j-1;\lambda_1,\lambda_2) + \lambda_2 c_{(N,4,j)}(N^2-j-1;\lambda_1,\lambda_2)$
2. $c_{(N,N^2+1,j)}(N^2-N-j;\lambda_1,\lambda_2) = c_{(N,N^2+2,j)}(N^2-N-j;\lambda_1,\lambda_2)$
3. $c_{(N,N^2-N+1,j)}(N^2-N+1-j;\lambda_1,\lambda_2) = (\lambda_1+\lambda_2)c_{(N,N^2+N+1,j)}(N^2-N-j;\lambda_1,\lambda_2)$
4. $c_{(N,(N+1)^2,j)}(N^2-N-j;\lambda_1,\lambda_2) = c_{(N,(N+1)^2-1),j}(N^2-N-j;\lambda_1,\lambda_2)$
5. $(\lambda_1+\lambda_2)c_{(N,N^2+N+1-y,j)}(N^2-N-j;\lambda_1,\lambda_2) = c_{(N,N^2-N+1-y,j)}(N^2-N+1-j;\lambda_1,\lambda_2)$
   $+\lambda_2 c_{(N,N^2+N+2-y,j)}(N^2-N-j;\lambda_1,\lambda_2),\ 0<y<N$
6. $(\lambda_1+\lambda_2)c_{(N,N^2+N+1+y,j)}(N^2-N-j;\lambda_1,\lambda_2) = c_{(N,N^2-N+1+y,j)}(N^2-N+1-j;\lambda_1,\lambda_2)$
   $+\lambda_1 c_{(N,N^2+N+y,j)}(N^2-N-j;\lambda_1,\lambda_2),\ 0<y<N$
7. $c_{(N,x^2+1,0)}(N^2-x;\lambda_1,\lambda_2) = c_{(N,x^2+2,0)}(N^2-x;\lambda_1,\lambda_2),\ 0<x<N$
8. $\lambda_2 c_{(N,x^2+1,j)}(N^2-x-j;\lambda_1,\lambda_2) + c_{(N,x^2+1,j-1)}(N^2-x-j+1;\lambda_1,\lambda_2)$
   $= \lambda_2 c_{(N,x^2+2,j)}(N^2-x-j;\lambda_1,\lambda_2) + \lambda_1 c_{(N,(x+1)^2+1,j-1)}(N^2-x-j;\lambda_1,\lambda_2),\ j\neq 0,\ 0<x<N$
9. $c_{(N,x^2+1,N^2-N)}(N-x;\lambda_1,\lambda_2) = \lambda_1 c_{(N,(x+1)^2+1,N^2-N)}(N-x-1;\lambda_1,\lambda_2),\ 0<x<N$
10. $(\lambda_1+\lambda_2)c_{(N,x^2+x+1-y,0)}(N^2-x;\lambda_1,\lambda_2) = c_{(N,x^2-x+1-y,0)}(N^2-x+1;\lambda_1,\lambda_2)$
    $+\lambda_2 c_{(N,x^2+x+2-y,0)}(N^2-x;\lambda_1,\lambda_2),\ 0<x<N,\ 0<y<x$
11. $c_{(N,x^2+x+1-y,j-1)}(N^2-x-j+1;\lambda_1,\lambda_2) + (\lambda_1+\lambda_2)c_{(N,x^2+x+1-y,j)}(N^2-x-j;\lambda_1,\lambda_2)$
    $= c_{(N,x^2-x+1-y,j)}(N^2-x-j+1;\lambda_1,\lambda_2) + \lambda_2 c_{(N,x^2+x+2-y,j)}(N^2-x-j;\lambda_1,\lambda_2)$
    $+\lambda_1 c_{(N,(x+1)^2+x+1-y,j-1)}(N^2-x-j;\lambda_1,\lambda_2),\ j\neq 0,\ 0<x<N,\ 0<y<x$
12. $c_{(N,x^2+x+1-y,N^2-N)}(N-x;\lambda_1,\lambda_2) = \lambda_1 c_{(N,(x+1)^2+x+1-y,N^2-N)}(N-x-1;\lambda_1,\lambda_2),\ 0<x<N,\ 0<y<x$
13. $c_{(N,x^2-x+1,0)}(N^2-x+1;\lambda_1,\lambda_2) = (\lambda_1+\lambda_2)c_{(N,x^2+x+1,0)}(N^2-x;\lambda_1,\lambda_2),\ 0<x<N$

14. $c_{(N,x^2+x+1,j-1)}(N^2-x-j+1;\lambda_1,\lambda_2)+(\lambda_1+\lambda_2)c_{(N,x^2+x+1,j)}(N^2-x-j;\lambda_1,\lambda_2)$
$= c_{(N,x^2-x+1,j)}(N^2-x-j+1;\lambda_1,\lambda_2)+\lambda_1 c_{(N,(x+1)^2+x+1,j-1)}(N^2-x-j;\lambda_1,\lambda_2)$
$+\lambda_2 c_{(N,(x+1)^2+x+3,j-1)}(N^2-x-j;\lambda_1,\lambda_2),\ j\neq 0,\ 0<x<N$
15. $c_{(N,x^2+x+1,N^2-N)}(N-x;\lambda_1,\lambda_2)=\lambda_1 c_{(N,(x+1)^2+x+1,N^2-N)}(N-x-1;\lambda_1,\lambda_2)$
$+\lambda_2 c_{(N,(x+1)^2+x+3,N^2-N)}(N-x-1;\lambda_1,\lambda_2),\ 0<x<N$
16. $(\lambda_1+\lambda_2)c_{(N,x^2+x+1+y,0)}(N^2-x;\lambda_1,\lambda_2)=c_{(N,x^2-x+1+y,0)}(N^2-x+1;\lambda_1,\lambda_2)$
$+\lambda_1 c_{(N,x^2+x+y,0)}(N^2-x;\lambda_1,\lambda_2),\ 0<x<N,\ 0<y<x$
17. $c_{(N,x^2+x+1+y,j-1)}(N^2-x-j+1;\lambda_1,\lambda_2)+(\lambda_1+\lambda_2)c_{(N,x^2+x+1+y,j)}(N^2-x-j;\lambda_1,\lambda_2)$
$= c_{(N,x^2-x+1+y,j)}(N^2-x-j+1;\lambda_1,\lambda_2)+\lambda_1 c_{(N,x^2+x+y,j)}(N^2-x-j;\lambda_1,\lambda_2)$
$+\lambda_2 c_{(N,(x+1)^2+x+y+3,j-1)}(N^2-x-j;\lambda_1,\lambda_2),\ j\neq 0,\ 0<x<N,\ 0<y<x$
18. $c_{(N,x^2+x+1+y,N^2-N)}(N-x;\lambda_1,\lambda_2)=\lambda_2 c_{(N,(x+1)^2+x+y+3,N^2-N)}(N-x-1;\lambda_1,\lambda_2),\ 0<x<N,\ 0<y<x$
19. $c_{(N,(x+1)^2,0)}(N^2-x;\lambda_1,\lambda_2)=c_{(N,(x+1)^2-1,0)}(N^2-x;\lambda_1,\lambda_2),\ 0<x<N$
20. $\lambda_1 c_{(N,(x+1)^2,j)}(N^2-x-j;\lambda_1,\lambda_2)+c_{(N,(x+1)^2,j-1)}(N^2-x-j+1;\lambda_1,\lambda_2)$
$=\lambda_1 c_{(N,(x+1)^2-1,j)}(N^2-x-j;\lambda_1,\lambda_2)+\lambda_2 c_{(N,(x+2)^2,j-1)}(N^2-x-j;\lambda_1,\lambda_2),$
$j\neq 0,\ 0<x<N$
21. $c_{(N,(x+1)^2,N^2-N)}(N-x;\lambda_1,\lambda_2)=\lambda_2 c_{(N,(x+2)^2,N^2-N)}(N-x-1;\lambda_1,\lambda_2),\ 0<x<N$

Proof (Sketch). Follows from Theorem 3 by substituting the solutions for the steady state probabilities into each of the GBEs, as in the proof of the theorem, and equating coefficients of ${\lambda_0}^j$ for all relevant j. □

Further, we state the following results regarding the polynomials for $N \geq 3$:

- $c_{(N,i,N^2-N)}(0;\lambda_1,\lambda_2)=1,\ N^2+1\leq i\leq (N+1)^2$;
- $c_{(N,1,0)}(N^2;\lambda_1,\lambda_2)=(\lambda_1+\lambda_2)^{N^2}$; and
- $c_{(N,1,N^2-N)}(N;\lambda_1,\lambda_2)={\lambda_1}^N+{\lambda_2}^N$.

4.5 Illustration of the General Form for $N = 3$

The general form has been validated from results obtained by Matlab. This section illustrates how the solution for $N = 3$, (see Appendix A of [10] which has Matlab results for $N = 1$ to 4), is mapped to the general form. For $N = 3$ the numerator polynomials are given by, for $1 \leq i \leq 16$,

$$C_{(3,i)}={\lambda_0}^{u(i)}{\lambda_1}^{3+u(i)-v(i)}{\lambda_2}^{3+v(i)-u(i)}\sum_{j=0}^{6}{\lambda_0}^j c_{(3,i,j)}(9-u(i)-j;\lambda_1,\lambda_2)$$

When $i = 1$, $u(i) = v(i) = 0$, and we obtain:

$$C_{(3,1)}={\lambda_1}^3{\lambda_2}^3\sum_{j=0}^{6}{\lambda_0}^j c_{(3,1,j)}(9-j;\lambda_1,\lambda_2)$$

Comparing this equation to the result for $C_{(3,1)}$ (see equation (A.16), p42 of [10]) we see that the general form is satisfied, with

$$
\begin{aligned}
c_{(3,1,6)}(3;\lambda_1,\lambda_2) &= {\lambda_1}^3 + {\lambda_2}^3 \\
c_{(3,1,5)}(4;\lambda_1,\lambda_2) &= 6{\lambda_1}^4 + 8{\lambda_1}^3\lambda_2 + 8\lambda_1{\lambda_2}^3 + 6{\lambda_2}^4 \\
c_{(3,1,4)}(5;\lambda_1,\lambda_2) &= 15{\lambda_1}^5 + 37{\lambda_1}^4\lambda_2 + 30{\lambda_1}^3{\lambda_2}^2 + 30{\lambda_1}^2{\lambda_2}^3 + 37\lambda_1{\lambda_2}^4 + 15{\lambda_2}^5 \\
c_{(3,1,3)}(6;\lambda_1,\lambda_2) &= 20\,{\lambda_1}^6 + 72{\lambda_1}^5\lambda_2 + 110{\lambda_1}^4{\lambda_2}^2 + 118{\lambda_1}^3{\lambda_2}^3 + 110{\lambda_1}^2{\lambda_2}^4 \\
&\quad + 72\lambda_1{\lambda_2}^5 + 20{\lambda_2}^6 \\
c_{(3,1,2)}(7;\lambda_1,\lambda_2) &= 15{\lambda_1}^7 + 74{\lambda_1}^6\lambda_2 + 165{\lambda_1}^5{\lambda_2}^2 + 230{\lambda_1}^4{\lambda_2}^3 + 230{\lambda_1}^3{\lambda_2}^4 \\
&\quad + 165{\lambda_1}^2{\lambda_2}^5 + 74\lambda_1{\lambda_2}^6 + 15{\lambda_2}^7 \\
c_{(3,1,1)}(8;\lambda_1,\lambda_2) &= 6{\lambda_1}^8 + 40{\lambda_1}^7\lambda_2 + 120{\lambda_1}^6{\lambda_2}^2 + 216{\lambda_1}^5{\lambda_2}^3 + 260{\lambda_1}^4{\lambda_2}^4 \\
&\quad + 216{\lambda_1}^3{\lambda_2}^5 + 120{\lambda_1}^2{\lambda_2}^6 + 40\lambda_1{\lambda_2}^7 + 6{\lambda_2}^8 \\
c_{(3,1,0)}(9;\lambda_1,\lambda_2) &= (\lambda_1+\lambda_2)^9
\end{aligned}
$$

Similar mappings can be obtained for the rest of the probabilities, identifying the remaining polynomials in λ_1 and λ_2 for $N = 3$.

We now illustrate Corollary 2. The second relationship, when $N = 3$, is given by $c_{(3,10,j)}(6-j;\lambda_1,\lambda_2) = c_{(3,11,j)}(6-j;\lambda_1,\lambda_2)$ for $0 \le j \le 6$. This is also confirmed by Matlab (see [10] pages 44 and 45).

4.6 Asymptotic Behaviour

Firstly we wish to check that the general form does correspond to our intuition for limiting behaviour. For this we consider the results when $\lambda_0 \to 0$. This case corresponds to no requests being made, and hence no activity in the fork-join. Thus we expect the probability of being in the initial marking to be one and all other probabilities to be zero. Secondly, we consider the case when there is no lack of requests, i.e. $\lambda_0 \to \infty$, corresponding to the GSPN in Fig.3.

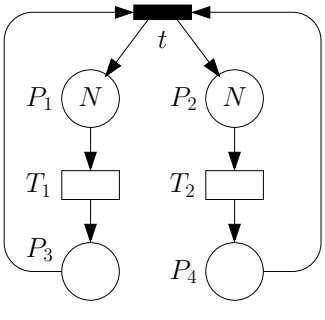


Fig. 3. The GSPN resulting from taking the limit as $\lambda_0 \to \infty$

State Probabilities when $\lambda_0 \to 0$. Firstly consider $i = 1$, when $\pi_{(N,1)} = \frac{C_{(N,1)}}{D_N}$ where

$$C_{(N,1)} = {\lambda_1}^N {\lambda_2}^N \sum_{j=0}^{N^2-N} {\lambda_0}^j c_{(N,1,j)}(N^2 - j; \lambda_1, \lambda_2)$$

As $\lambda_0 \to 0$ we only consider coefficients of ${\lambda_0}^0$, in both $C_{(N,1)}$ and D_N as all other terms will tend to zero. Since for all $i > 1$, all terms in $C_{(N,i)}$ have λ_0 as a factor (from equation (17)), both $C_{(N,1)}$ and D_N reduce to ${\lambda_1}^N {\lambda_2}^N c_{(N,1,0)}(N^2; \lambda_1, \lambda_2)$. Thus

$$\lim_{\lambda_0 \to 0} \pi_{(N,1)} = 1$$

Again, since for all $i > 1$, all terms in $C_{(N,i)}$ have λ_0 as a factor and D_N reduces to ${\lambda_1}^N {\lambda_2}^N c_{(N,1,0)}(N^2; \lambda_1, \lambda_2)$, we obtain

$$\lim_{\lambda_0 \to 0} \pi_{(N,i)} = 0,\ 1 < i \leq (N+1)^2$$

The results also hold for $N = 1, 2$, as can be seen by setting $\lambda_0 = 0$ in the equations for their respective probabilities. This is because for $\lambda_0 \ll min(\lambda_1, \lambda_2)$, the fork-join is much faster than the requests, and hence all the tokens spend most of their time in P_0. This confirms our intuition.

State Probabilities when $\lambda_0 \to \infty$. In this case, terms with the maximum degree of λ_0 will dominate. Hence we ignore other terms so that from equation (17) we obtain:

$$\lim_{\lambda_0 \to \infty} C_{(N,i)} = {\lambda_0}^{N^2-N+u(i)} {\lambda_1}^{N+u(i)-v(i)} {\lambda_2}^{N+v(i)-u(i)} c_{(N,i,N^2-N)}(N - u(i); \lambda_1, \lambda_2)$$

The degree of λ_0 is greatest when $u(i) = N$, which corresponds to the right most column of states, and occurs when $N^2 + 1 \leq i \leq (N+1)^2$. In this case the degree of λ_0 is N^2. From equation (18) the maximum degree of λ_0 in D_N is thus N^2. Hence all the probabilities where $1 \leq i \leq N^2$ tend to zero, as the degree of λ_0 in the denominator is greater than its degree in the numerator. For $N^2+1 \leq i \leq (N+1)^2$, $u(i) = N$ and $v(i) = i-N^2-1$ and therefore $C_{(N,i)}$ tends to ${\lambda_0}^{N^2} {\lambda_1}^{N^2+2N+1-i} {\lambda_2}^{i-N^2-1} c_{(N,i,N^2-N)}(0; \lambda_1, \lambda_2) = {\lambda_0}^{N^2} {\lambda_1}^{N^2+2N+1-i} {\lambda_2}^{i-N^2-1}$. Hence D_N tends to ${\lambda_0}^{N^2} \sum_{m=0}^{2N} {\lambda_1}^{2N-m} {\lambda_2}^m$, when considering only terms of the greatest degree in λ_0 and substituting $m = i - N^2 - 1$. Thus

$$\lim_{\lambda_0 \to \infty} \pi_{(N,i)} = \begin{cases} 0, & 1 \leq i \leq N^2 \\ \frac{{\lambda_1}^{N^2+2N+1-i} {\lambda_2}^{i-(N^2+1)}}{{\lambda_1}^{2N} \sum_{m=0}^{2N} {\lambda_1}^{-m} {\lambda_2}^m}, & N^2+1 \leq i \leq (N+1)^2 \end{cases}$$

$$= \begin{cases} 0, & 1 \leq i \leq N^2 \\ \frac{(\frac{\lambda_2}{\lambda_1})^{i-(N^2+1)}}{\sum_{m=0}^{2N} (\frac{\lambda_2}{\lambda_1})^m}, & N^2+1 \leq i \leq (N+1)^2 \end{cases}$$

When $N^2+1 \leq i \leq (N+1)^2$ and for $\lambda_1 \neq \lambda_2$, we obtain the following closed form expression by using the formula for geometric series:

$$\lim_{\lambda_0 \to \infty} \pi_{(N,i)} = \left(\frac{\lambda_2}{\lambda_1}\right)^{i-(N^2+1)} \frac{1-\frac{\lambda_2}{\lambda_1}}{1-\left(\frac{\lambda_2}{\lambda_1}\right)^{2N+1}} \tag{22}$$

Thus the probabilities of being in the right most column of states are a function of the ratio of rates in the fork-join, $r_{21} = \frac{\lambda_2}{\lambda_1}$, and moreover form a geometric progression with common ratio r_{21}. We can also express these probabilities as a function of the inverse ratio $r_{12} = \frac{\lambda_1}{\lambda_2}$ by multiplying the numerator and denominator of equation (22) by $\left(\frac{\lambda_1}{\lambda_2}\right)^{2N+1}$ and re-arranging:

$$\lim_{\lambda_0 \to \infty} \pi_{(N,i)} = \left(\frac{\lambda_1}{\lambda_2}\right)^{(N+1)^2-i} \frac{1-\frac{\lambda_1}{\lambda_2}}{1-\left(\frac{\lambda_1}{\lambda_2}\right)^{2N+1}} \tag{23}$$

As indicated in the introduction, closed form approximate solutions for the steady state probabilities were derived from numerical results in [4]. Equations (22) and (23) imply that these approximate solutions become exact as $\lambda_0 \to \infty$. Equations (22) and (23) also give rise to the following convergence results:

$$\lim_{\lambda_0, N \to \infty} \pi_{(N,N^2+1)} = 1 - \frac{\lambda_2}{\lambda_1},\ \lambda_1 > \lambda_2$$

$$\lim_{\lambda_0, N \to \infty} \pi_{(N,(N+1)^2)} = 1 - \frac{\lambda_1}{\lambda_2},\ \lambda_2 > \lambda_1$$

These results validate the approximate expressions in [4] when $\lambda_0 \to \infty$.

When $\lambda_1 = \lambda_2$, the sum becomes $2N+1$, giving rise to the intuitively appealing result where it is equiprobable to be in any of the states in which all the tokens are in the fork-join:

$$\lim_{\lambda_0 \to \infty} \pi_{(N,i)} = \frac{1}{2N+1}, \quad N^2+1 \leq i \leq (N+1)^2 \tag{24}$$

These results also hold for $N = 1, 2$, by just taking the coefficients of λ_0 for $N = 1$ and ${\lambda_0}^3$ for $N = 2$ in the equations for the appropriate probabilities for $N^2+1 \leq i \leq (N+1)^2$. For the other probabilities, the degree of λ_0 is higher in the denominator than the numerator, and hence they tend to zero.

Equations (22), (23) and (24) can also be corroborated by obtaining the families of RRGs, CTMCs and GBEs for the GSPN in Fig. 3, and solving for the probabilities. This turns out to be relatively straightforward.

These results correspond to all tokens spending nearly all their time in the fork-join when $\lambda_0 \gg min(\lambda_1, \lambda_2)$, i.e. the requests are fast enough to keep both processes at full capacity.

5 Conclusions

This paper has presented a parametric analysis of a cyclic Generalized Stochastic Petri Net (GSPN) with a fork-join subnet. The GSPN uses symbolic values for each of the firing rates: λ_0 for the request rate; and λ_1 and λ_2 for the processing rates in the two branches of the fork-join. Our first key result is a theorem defining the parametric reduced reachability graph, RRG_N, which we proved to have one strongly connected component and $(N+1)^2$ states. We transformed RRG_N to a parametric Markov Chain, and proved it to be irreducible. This allowed us to derive the set of $(N+1)^2$ global balance equations as a scheme of 11 equations relating the steady state probabilities. The solution of these equations for arbitrary N is a difficult problem. We presented solutions for the steady state probabilities for $N = 1, 2$ and then stated and proved a theorem for the general form of the solution for $N \geq 3$. The theorem demonstrates that the probabilities can be expressed as a ratio of polynomials in λ_0, the coefficients of which are polynomials in λ_1 and λ_2. The general form of the solution also holds for $N = 1$, but not for $N = 2$. Curiously, when $N = 2$, the general form holds if N^2 is replaced by $N^2 - 1$. The general form is illustrated for the case $N = 3$. We also showed that a scheme of 21 equations exist that provide simple relationships between the polynomials in λ_1 and λ_2. Finally we provided some asymptotic results. When λ_0 tends to 0, the probability of the system being idle tends to 1 as expected, providing a useful check on the result. However, when λ_0 tends to infinity, both processes operate at full capacity, and we have obtained expressions for the probabilities of being in each of the possible states for arbitrary N. In all but the right most column of states, this probability is zero. The probability of being in a state in the right most column follows a geometric progression in the ratio of the rates for each branch (i.e. $\frac{\lambda_1}{\lambda_2}$), except when $\lambda_1 = \lambda_2$ when all states are equiprobable. The expressions also allow us to derive convergence results for large N for the probability of one process being ahead of the other by N. Importantly, these asymptotic results validate the approximate solutions obtained empirically in [4] and imply that the approximations become exact as $\lambda_0 \to \infty$.

The general form of the solution for the steady state probabilities demonstrates that they are very complex, and generates a family of polynomials that are worthy of further study in their own right. For example, the scheme of 21 equations, relating the different polynomials in λ_1 and λ_2, has the potential to provide an alternative method of computing the steady state probabilities. Further, it will be interesting to study their convergence properties as N grows large as a prelude to proving that the steady state probabilities converge for (not so) large N, a property for which we have strong empirical evidence [4, 9]. Given the results of this paper and those in [4], we are interested in deriving performance measures, such as the mean request acceptance rate, process utilisation, mean process idle time, average buffer sizes and the lead/lag of the two processes that can be obtained from the steady state probabilities [18]. Future work also includes a major generalization to two parameters by considering any number of parallel processes in the fork-join, as well as the capacity constraint N.

Acknowledgements. The authors are grateful for the constructive comments of the anonymous referees.

References

1. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley and Sons, Chichester (1995)
2. Baccelli, F., Massey, W.A., Towsley, D.: Acyclic fork-join queuing networks. Journal of the ACM 36(3), 615–642 (1989)
3. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets - An Introduction to the Theory, 2nd edn. Vieweg-Verlag (2002)
4. Billington, J., Gallasch, G.E.: Closed Form Approximations for Steady State Probabilities of a Controlled Fork-Join Network. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 420–435. Springer, Heidelberg (2010)
5. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
6. Flatto, L.: Two Parallel Queues Created by Arrivals with Two Demands II. SIAM Journal of Applied Mathematics 45, 861–878 (1985)
7. Flatto, L., Hahn, S.: Two Parallel Queues Created by Arrivals with Two Demands I. SIAM Journal of Applied Mathematics 44, 1041–1053 (1984)
8. Freiheit, J., Billington, J.: New Developments in Closed-Form Computation for GSPN Aggregation. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 471–490. Springer, Heidelberg (2003)
9. Gallasch, G.E., Billington, J.: A Study of the Convergence of Steady State Probabilities in a Closed Fork-Join Network. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 143–157. Springer, Heidelberg (2010)
10. Gallasch, G.E., Billington, J.: Exploring Parametric Representation and Aggregation of Closed Fork-Join Subnets. Tech. Rep. CSEC-40, Computer Systems Engineering Centre Report Series, University of South Australia (March 2010)
11. Harrison, P., Zertal, S.: Queueing models of RAID systems with maxima of waiting times. Performance Evaluation 64, 664–689 (2007)
12. Kim, C., Agrawala, A.K.: Analysis of the Fork-Join Queue. IEEE Transactions on Computers 38(2), 250–255 (1989)
13. Lilith, N., Billington, J., Freiheit, J.: Approximate Closed-Form Aggregation of a Fork-Join Structure in Generalised Stochastic Petri Nets. In: Proc. 1st Int. Conference on Performance Evaluation Methodologies and Tools. International Conference Proceedings Series, vol. 180, Article 32, 10 pages. ACM Press, New York (2006)
14. Liu, Y.C., Perros, H.G.: A Decomposition Procedure for the Analysis of a Closed Fork/Join Queueing System. IEEE Transactions on Computers 40(3), 365–370 (1991)
15. Lui, J.C.S., Muntz, R.R., Towsley, D.: Computing Performance Bounds of Fork-Join Parallel Programs under a Multiprocessing Environment. IEEE Transactions on Parallel and Distributed Systems 9(3), 295–311 (1998)
16. Makowski, A., Varma, S.: Interpolation Approximations for Symmetric Fork-Join Queues. Performance Evaluation 20, 145–165 (1994)
17. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)

18. Neale, G.: On Performance Analysis of On-Demand Parallel Manufacturing Systems Modelled by Generalised Stochastic Petri Nets. Honours thesis, University of South Australia, School of Electrical and Information Engineering (2010)
19. Nelson, R., Tantawi, A.N.: Approximate Analysis of Fork/Join Synchronization in Parallel Queues. IEEE Transactions on Computers 37(6), 739–743 (1988)
20. Nelson, R., Towsley, D., Tantawi, A.N.: Performance Analysis of Parallel Processing Systems. IEEE Transactions on Software Engineering 14(4), 532–540 (1988)
21. Varki, E.: Mean value technique for closed fork-join networks. In: SIGMETRICS 1999: Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling Of Computer Systems, pp. 103–112 (1999)
22. Varki, E.: Response Time Analysis of Parallel Computer and Storage Systems. IEEE Transactions on Parallel and Distributed Systems 12(11), 1146–1161 (2001)

Synthesis and Analysis of Product-Form Petri Nets

S. Haddad[1], J. Mairesse[2], and H.-T. Nguyen[2]

[1] ENS Cachan, LSV, CNRS UMR 8643, INRIA, Cachan, France
haddad@lsv.ens-cachan.fr
[2] Université Paris 7, LIAFA, CNRS UMR 7089, Paris, France
{mairesse,ngthach}@liafa.jussieu.fr

Abstract. For a large Markovian model, a "product form" is an explicit description of the steady-state behaviour which is otherwise generally untractable. Being first introduced in queueing networks, it has been adapted to Markovian Petri nets. Here we address three relevant issues for product-form Petri nets which were left fully or partially open: (1) we provide a sound and complete set of rules for the synthesis; (2) we characterise the exact complexity of classical problems like reachability; (3) we introduce a new subclass for which the normalising constant (a crucial value for product-form expression) can be efficiently computed.

Keywords: Petri nets, product-form, synthesis, complexity analysis, reachability, normalising constant.

1 Introduction

Product-form for stochastic models. Markovian models of discrete events systems are powerful formalisms for modelling and evaluating the performances of such systems. The main goal is the equilibrium performance analysis. It requires to compute the stationary distribution of a continuous time Markov process derived from the model. Unfortunately the potentially huge (sometimes infinite) state space of the models often prevents the modeller from computing explicitly this distribution. To cope with the issue, one can forget about exact solutions and settle for approximations, bounds, or even simulations. The other possibility is to focus on subclasses for which some kind of explicit description is indeed possible. In this direction, the most efficient and satisfactory approach may be the *product-form* method: for a model composed of modules, the stationary probability of a global state may be expressed as a product of quantities depending only on local states divided by a *normalising constant*.

Such a method is applicable when the interactions between the modules are "weak". This is the case for queueing networks where the interactions between queues are described by a random routing of clients. Various classes of queueing networks with product-form solutions have been exhibited [11, 3, 12]. Moreover efficient algorithms have been designed for the computation of the normalising constant [18].

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 288–307, 2011.

Product-form Petri nets. Due to the explicit modelling of competition and synchronisation, the Markovian Petri nets formalism [1] is an attractive modelling paradigm. Similarly to queueing networks, product-form Markovian Petri Nets were introduced to cope with the combinatorial explosion of the state space. Historically, works started with purely behavioural properties (i.e. by an analysis of the reachability graph) as in [13], and then progressively moved to more and more structural characterisations [14, 10]. Building on the work of [10], the authors of [9] establish the first purely structural condition for which a product form exists and propose a polynomial time algorithm to check for the condition, see also [15] for an alternative characterisation. These nets are called Π^2-nets.

Open issues related to product-form Petri nets

- From a modelling point of view, it is more interesting to design specific types of Petri nets by modular constructions rather than checking a posteriori whether a net satisfies the specification. For instance, in [7], a sound and complete set of rules is proposed for the synthesis of live and bounded free-choice nets. Is it possible to get an analog for product-form Petri nets?
- From a qualitative analysis point of view, it is interesting to know the complexity of classical problems (reachability, coverability, liveness, etc.) for a given subclass of Petri nets and to compare it with that of general Petri nets. For product-form Petri nets, partial results were presented in [9] but several questions were left open. For instance, the reachability problem is PSPACE-complete for safe Petri nets but in safe product-form Petri nets it is only proved to be NP-hard in [9].
- From a quantitative analysis point of view, an important and difficult issue is the computation of the normalising constant. Indeed, in product-form Petri nets, one can directly compute relative probabilities (e.g. available versus unavailable service), but determining absolute probabilities requires to compute the normalising constant (i.e. the sum over reachable states of the relative probabilities). In models of queueing networks, this can be efficiently performed using dynamic programming. In Petri nets, it has been proved that the efficient computation is possible when the linear invariants characterise the set of reachable markings [6]. Unfortunately, all the known subclasses of product-form nets that fulfill this characterisation are models of queueing networks!

Our contribution. Here we address the three above issues. In Section 3, we provide a set of sound and complete rules for generating any Π^2-net. We also use these rules for transforming a general Petri net into a related product-form Petri net. In Section 4, we solve relevant complexity issues. More precisely, we show that the reachability and liveness problems are PSPACE-complete for safe product-form nets and that the coverability problem is EXPSPACE-complete for general product-form nets. From these complexity results, we conjecture that the problem of computing the normalising constant does not admit an efficient solution for the general class of product-form Petri nets. However, in Section 5, we introduce a large subclass of product-form Petri nets, denoted Π^3-nets,

for which the normalising constant can be efficiently computed. We emphasise that contrary to all subclasses related to queueing networks, Π^3-nets may admit *spurious* markings (i.e. that fufill the invariants while being unreachable).

The above results may change our perspective on product-form Petri nets. It is proved in [15] that the intersection of free-choice and product-form Petri nets is the class of Jackson networks [11]. This may suggest that the class of product-form Petri nets is somehow included in the class of product-form queueing networks. In the present paper, we refute this belief in two ways. First by showing that some classical problems are as complex for product-form Petri nets as for general Petri nets whereas they become very simple for product-form queueing networks. Second by exhibiting the class of Π^3-nets, see the above discussion.

A version of the present paper including proofs can be found on arXiv.

Notations. We often denote a vector $u \in \ ^S$ by $\sum_s u(s)s$. The *support* of vector u is the subset $S' \equiv \{s \in S \mid u(s) \neq 0\}$.

2 Petri Nets, Product-Form Nets, and Π^2-Nets

Definition 2.1 (Petri net). *A* Petri net *is a 5-tuple* $\mathcal{N} = (P, T, W^-, W^+, m_0)$ *where:*

- P *is a finite set of* places*;*
- T *is a finite set of* transitions*, disjoint from* P*;*
- W^-*, resp.* W^+*, is a* $P \times T$ *matrix with coefficients in* $\ ;$
- $m_0 \in \ ^P$ *is the* initial marking.

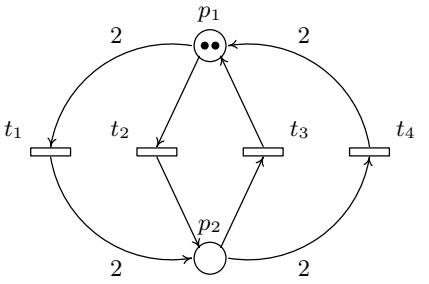
Below, we also call *Petri net* the unmarked quadruple (P, T, W^-, W^+). The presence or absence of a marking will depend on the context.

A Petri net is represented in Figure 1. The following graphical conventions are used: places are represented by circles and transitions by rectangles. There is an arc from $p \in P$ to $t \in T$ (resp. from $t \in T$ to $p \in P$) if $W^+(p,t) > 0$ (resp. $W^-(p,t) > 0$), and the weight $W^+(p,t)$ (resp. $W^-(p,t)$) is written above the corresponding arc except when it is equal to 1 in which case it is omitted. The initial marking is materialised: if $m_0(p) = k$, then k tokens are drawn inside the circle p. Let $P' \subset P$ and m be a marking then $m(P')$ is defined by $m(P') \equiv \sum_{p \in P'} m(p)$.

The matrix $W = W^+ - W^-$ is the *incidence matrix* of the Petri net. The *input bag* ${}^\bullet t$ (resp. *output bag* $t^\bullet$) of the transition t is the column vector of W^- (resp. W^+) indexed by t. For a place p, we define ${}^\bullet p$ and $p^\bullet$ similarly. A *T-semi-flow* (resp. *S-semi-flow*) is a $\mathbb{Q}$-valued vector v such that $W.v = 0$ (resp. $v.W = 0$).

A *symmetric* Petri net is a Petri net such that: $\forall t \in T,\ \exists t^- \in T,\ \ {}^\bullet t = (t^-)^\bullet, t^\bullet = {}^\bullet t^-$. A *state machine* is a Petri net such that: $\forall t \in T,\ |{}^\bullet t| = |t^\bullet| = 1$.

Definition 2.2 (Firing rule). *A transition* t *is* enabled *by the marking* m *if* $m \geq {}^\bullet t$ *(denoted by* $m \xrightarrow{t}$*); an enabled transition* t *may* fire *which transforms the marking* m *into* $m - {}^\bullet t + t^\bullet$*, denoted by* $m \xrightarrow{t} m' = m - {}^\bullet t + t^\bullet$.



$$W = \begin{pmatrix} -2 & -1 & 1 & 2 \\ 2 & 1 & -1 & -2 \end{pmatrix} .$$
$$m_0 = (2, 0) .$$

Fig. 1. Petri net

A marking m' is *reachable* from the marking m if there exists a *firing sequence* $\sigma = t_1 \dots t_k$ $(k \geq 0)$ and a sequence of markings $m_1, \dots, m_{k-1}$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \cdots \xrightarrow{t_{k-1}} m_{k-1} \xrightarrow{t_k} m'$. We write in a condensed way: $m \xrightarrow{\sigma} m'$.

We denote by $\mathcal{R}(m)$ the set of markings which are reachable from the marking m. The *reachability graph* of a Petri net with initial marking m_0 is the directed graph with nodes $\mathcal{R}(m_0)$ and arcs $\{(m, m') | \exists t \in T : m \xrightarrow{t} m'\}$.

Given $(\mathcal{N}, m_0)$ and m_1, the *reachability problem* is to decide if $m_1 \in \mathcal{R}(m_0)$, and the *coverability problem* is to decide if $\exists m_2 \in \mathcal{R}(m_0), m_2 \geq m_1$.

A Petri net $(\mathcal{N}, m_0)$ is *live* if every transition can always be enabled again, that is: $\forall m \in \mathcal{R}(m_0), \forall t \in T,\ \exists m' \in \mathcal{R}(m),\ m' \xrightarrow{t}$. A Petri net $(\mathcal{N}, m_0)$ is bounded if $\mathcal{R}(m_0)$ is finite. It is *safe* or *1-bounded* if: $\forall m \in \mathcal{R}(m_0),\ m(p) \leq 1$.

2.1 Product-Form Petri Nets

There exist several ways to define timed models of Petri nets, see [2]. We consider the model of Markovian Petri nets with *race policy*. Roughly, with each enabled transition is associated a "countdown clock" whose positive initial value is set at random according to an exponential distribution whose rate depends on the transition. The first transition to reach 0 fires, which may enable new transitions and start new clocks.

Definition 2.3 (Markovian PN). *A* Markovian Petri net (with race policy) *is a Petri net equipped with a set of* rates $(\mu_t)_{t \in T}$, $\mu_t \in \ _+ \setminus \{0\}$. *The firing time of an enabled transition t is exponentially distributed with parameter μ_t. The marking evolves as a continuous-time jump Markov process with state space $\mathcal{R}(m_0)$ and infinitesimal generator $Q = (q_{m,m'})_{m,m' \in \mathcal{R}(m_0)}$, given by:*

$$\forall m,\ \forall m' \neq m,\ q_{m,m'} = \sum_{t : m \xrightarrow{t} m'} \mu_t, \qquad \forall m,\ q_{m,m} = - \sum_{m' \neq m} q_{m,m'} . \tag{2.1}$$

W.l.o.g., we assume that there is no transition t such that ${}^\bullet t = t^\bullet$. Indeed, the firing of such a transition does not modifiy the marking, so its removal does not modify the infinitesimal generator. We also assume that $({}^\bullet t_1, t_1^\bullet) \neq ({}^\bullet t_2, t_2^\bullet)$ for

all transitions $t_1 \neq t_2$. Indeed, if it is not the case, the two transitions may be replaced by a single one with the summed rate.

An *invariant measure* is a non-trivial solution ν to the *balance equations*: $\nu Q = 0$. A *stationary measure (distribution)* π is an invariant probability measure: $\pi Q = 0$, $\sum_m \pi(m) = 1$.

Definition 2.4 (Product-form PN). *A Petri net is a* product-form Petri net *if for all rates $(\mu_t)_{t\in T}$, the corresponding Markovian Petri net admits an invariant measure ν satisfying:*

$$\exists (u_p)_{p\in P}, u_p \in \ _+, \quad \forall m \in \mathcal{R}(m_0), \qquad \nu(m) = \prod_{p\in P} u_p^{m_p} \, . \tag{2.2}$$

The existence of ν satisfying (2.2) implies that the marking process is irreducible (in other words, the reachability graph is strongly connected). In (2.2), the mass of the measure, i.e. $\nu(\mathcal{R}(m_0)) = \sum_m \nu(m)$, may be either finite or infinite. For a bounded Petri net, the mass is always finite. But for an unbounded Petri net, the typical situation will be as follows: structural conditions on the Petri net will ensure that the Petri net is a product-form one. Then, for some values of the rates, ν will have an infinite mass, and, for others, ν will have a finite mass. In the first situation, the marking process will be either transient or recurrent null (unstable case). In the second situation, the marking process will be positive recurrent (stable or ergodic case).

When the mass is finite, we call $\nu(\mathcal{R}(m_0))$ the *normalising constant*. The probability measure $\pi(\cdot) = \nu(\mathcal{R}(m_0))^{-1}\nu(\cdot)$ is the unique stationary measure of the marking process. Computing explicitly the normalising constant is an important issue, see Section 5.

The goal is now to get sufficient conditions for a Petri net to be of product-form. To that purpose, we introduce three notions: *weak reversibility*, *deficiency*, and *witnesses*.

Let (N, m_0) be a Petri net. The set of *complexes* is defined by $\mathcal{C} = \{{}^\bullet t \mid t \in T\} \cup \{t^\bullet \mid t \in T\}$. The *reaction graph* is the directed graph whose set of nodes is $\mathcal{C}$ and whose set of arcs is $\{({}^\bullet t, t^\bullet) | t \in T\}$.

Definition 2.5 (Weak reversibility: Π-nets). *A Petri net is* weakly reversible (WR) *if every connected component of its reaction graph is strongly connected. Weakly reversible Petri nets are also called* Π-nets.

The notion and the name "WR" come from the chemical literature. In the Petri net context, it was introduced in [4, Assumption 3.2] under a different name and with a slightly different but equivalent formulation. WR is a strong constraint. It should not be confused with the classical notion of "reversibility" (the marking graph is strongly connected). In particular, WR implies reversibility! Observe that all symmetric Petri nets are WR.

The notion of deficiency is due to Feinberg [8].

Definition 2.6 (Deficiency). *Consider a Petri net with incidence matrix W and set of complexes $\mathcal{C}$. Let ℓ be the number of connected components of the reaction graph. The* deficiency *of the Petri net is defined by: $|\mathcal{C}| - \ell - rank(W)$.*

The notion of witnesses appears in [9].

Definition 2.7 (Witness). *Let c be a complex. A* witness *of c is a vector $wit(c) \in \mathbb{Q}^P$ such that for all transition t:*

$$\begin{cases} wit(c) \cdot W(t) = -1 & \text{if } {}^\bullet t = c \\ wit(c) \cdot W(t) = 1 & \text{if } t^\bullet = c \\ wit(c) \cdot W(t) = 0 & \text{otherwise}\,, \end{cases}$$

where $W(t)$ denotes the column vector of W indexed by t.

Examples. Consider the Petri net of Figure 1. First, it is WR. Indeed, the set of complexes is $\mathcal{C} = \{p_1, p_2, 2p_1, 2p_2\}$ and the reaction graph is:

$$p_1 \leftrightarrow p_2\,,\ 2p_1 \leftrightarrow 2p_2\,,$$

with two connected components which are strongly connected. Second, the deficiency is 1 since $|\mathcal{C}| = 4$, $\ell = 2$, and $\mathrm{rank}(W) = 1$. Last, one can check that none of the complexes admit a witness.

Consider now the Petri net of Figure 3. It is WR and it has deficiency 0.

Proposition 2.8 (deficiency 0 $\iff$ witnesses, in [15, Prop. 3.9]). *A Petri net admits a witness for each complex iff it has deficiency 0.*

Next Theorem is a combination of Feinberg's Deficiency zero Theorem [8] and Kelly's Theorem [12, Theorem 8.1]. (It is proved under this form in [15, Theorem 3.8].)

Theorem 2.9 (WR + deficiency 0 $\implies$ product-form). *Consider a Markovian Petri net with rates $(\mu_t)_{t\in T}$, $\mu_t > 0$, and assume that the underlying Petri net is WR and has deficiency 0. Then there exists $(u_p)_{p\in P}$, $u_p > 0$, satisfying the equations:*

$$\forall c \in \mathcal{C}, \qquad \prod_{p:c_p\neq 0} u_p^{c_p} \sum_{t:{}^\bullet t = c} \mu_t = \sum_{t:t^\bullet = c} \mu_t \prod_{p:{}^\bullet t_p \neq 0} u_p^{{}^\bullet t_p}\,. \tag{2.3}$$

The marking process has an invariant measure ν s.t.: $\forall m,\ \nu(m) = \prod_{p\in P} u_p^{m_p}$.

Checking the WR, computing the deficiency, determining the witnesses, and solving the equations (2.3), all of these operations can be performed in polynomial-time, see [9, 15].

Summing up the above, it seems worth to isolate and christen the class of nets which are WR and have deficiency 0. We adopt the terminology of [9].

Definition 2.10 (Π^2-net). *A* Π^2-net *is a Petri net which is WR and has deficiency 0.*

3 Synthesis and Regulation of Π^2-Nets

The reaction graph, defined in Section 2.1, may be viewed as a Petri net (state machine). Let us formalise this observation. The *reaction Petri net* of $\mathcal{N}$ is the Petri net $\mathcal{A} = (\mathcal{C}, T, \overline{W}^-, \overline{W}^+)$, with for every $t \in T$:
- $\overline{W}^-({}^\bullet t, t) = 1$ and $\forall u \neq {}^\bullet t,\ \overline{W}^-(u,t) = 0$
- $\overline{W}^+(t^\bullet, t) = 1$ and $\forall u \neq t^\bullet,\ \overline{W}^+(u,t) = 0$

3.1 Synthesis

In this subsection, we consider unmarked nets. We define three rules that generate all the Π^2-nets. The first rule adds a strongly connected state machine.

Definition 3.1 (State-machine insertion). *Let $\mathcal{N} = (P_\mathcal{N}, T_\mathcal{N}, W^-_\mathcal{N}, W^+_\mathcal{N})$ be a net and $\mathcal{M} = (P_\mathcal{M}, T_\mathcal{M}, W^-_\mathcal{M}, W^+_\mathcal{M})$ be a strongly connected state machine disjoint from $\mathcal{N}$. The rule* `S-add` *is always applicable and $\mathcal{N}' =$* `S-add`*$(\mathcal{N}, \mathcal{M})$ is defined by:*
- $P' = P_\mathcal{N} \sqcup P_\mathcal{M}$, $T' = T_\mathcal{N} \sqcup T_\mathcal{M}$;
- $\forall p \in P_\mathcal{N},\ \forall t \in T_\mathcal{N},\ W'^-(p,t) = W^-_\mathcal{N}(p,t),\ W'^+(p,t) = W^+_\mathcal{N}(p,t)$;
- $\forall p \in P_\mathcal{M},\ \forall t \in T_\mathcal{M},\ W'^-(p,t) = W^-_\mathcal{M}(p,t),\ W'^+(p,t) = W^+_\mathcal{M}(p,t)$;
- *All other entries of W'^- and W'^+ are null.*

The second rule consists in substituting to a complex c the complex $c + \lambda p$. However in order to be applicable some conditions must be fulfilled. The first condition requires that $c(p) + \lambda$ is non-negative. The second condition ensures that the substitution does not modify the reaction graph. The third condition preserves deficiency zero. Observe that the third condition can be checked in polynomial time, indeed it amounts to solving a system of linear equations in $\mathbb{Q}$ for every complex.

Definition 3.2 (Complex update). *Let $\mathcal{N} = (P, T, W^-, W^+)$ be a Π^2-net, c be a complex of $\mathcal{N}$, $p \in P$, $\lambda \in\ ^*$. The rule* `C-update` *is applicable when:*
1. *$\lambda + c(p) \geq 0$;*
2. *$c + \lambda p$ is not a complex of $\mathcal{N}$;*
3. *For every complex c' there exists a witness $wit(c')$ s.t. $wit(c')(p) = 0$.*

The resulting net $\mathcal{N}' =$ `C-update`*$(\mathcal{N}, c, p, \lambda)$ is defined by:*
- $P' = P$, $T' = T$;
- $\forall t \in T$ *s.t.* $W^-(t) \neq c,\ W'^-(t) = W^-(t)$, $\forall t \in T$ *s.t.* $W^-(t) = c,\ W'^-(t) = c + \lambda p$
- $\forall t \in T$ *s.t.* $W^+(t) \neq c,\ W'^+(t) = W^-(t)$, $\forall t \in T$ *s.t.* $W^+(t) = c,\ W'^+(t) = c + \lambda p$.

The last rule "cleans" the net by deleting an isolated place. We call this operation `P-delete`.

Definition 3.3 (Place deletion). *Let $\mathcal{N} = (P, T, W^-, W^+)$ be a net and let p be an isolated place of $\mathcal{N}$, i.e. $W^-(p) = W^+(p) = 0$. Then the rule* `P-delete` *is applicable and $\mathcal{N}' =$* `P-delete`*$(\mathcal{N}, p)$ is defined by:*
- $P' = P \setminus \{p\}$, $T' = T$;

– $\forall q \in P',\ W'^{-}(q) = W^{-}(q),\ W'^{+}(q) = W^{+}(q)$.

Proposition 3.4 shows the interest of the rules for synthesis of Π^2-nets.

Proposition 3.4 (Soundness and Completeness). *Let $\mathcal{N}$ be a Π^2-net.*
– *If a rule* `S-add`*,* `C-update` *or* `P-delete` *is applicable on $\mathcal{N}$ then the resulting net is still a Π^2-net.*
– *The net $\mathcal{N}$ can be obtained by successive applications of the rules* `S-add`*,* `C-update`*,* `P-delete` *starting from the empty net.*

We illustrate the synthesis process using our rules on the net numbered 5 in Figure 2. We have also indicated on the right upper part of this figure, the four complexes and their witnesses. Since the reaction Petri graph of this net has two state machines, we start by creating it using twice the insertion of a state machine (net 1). Then we add the place p_1 (a particular state machine). We update the complex c_1 (the single one where p_1 appears in the original net) by adding $3p_1$ (net 2). The new complex cannot appear elsewhere due to the presence of c_1. Iterating this process, we obtain the net 3. Observe that this net is a fusion (via T the set of transitions) of the original net and its reaction Petri net. We now iteratively update the complexes. The net 4 is the result of transforming $c_1 + 3p_1$ into $3p_1$. This transformation is applicable since all the complexes are witnessed by witnesses of the original net. For instance, $c_1 + 3p_1$ is witnessed by $(1/3)p_1$. Once c_1 is isolated, we delete it. Iterating this process yields the original net.

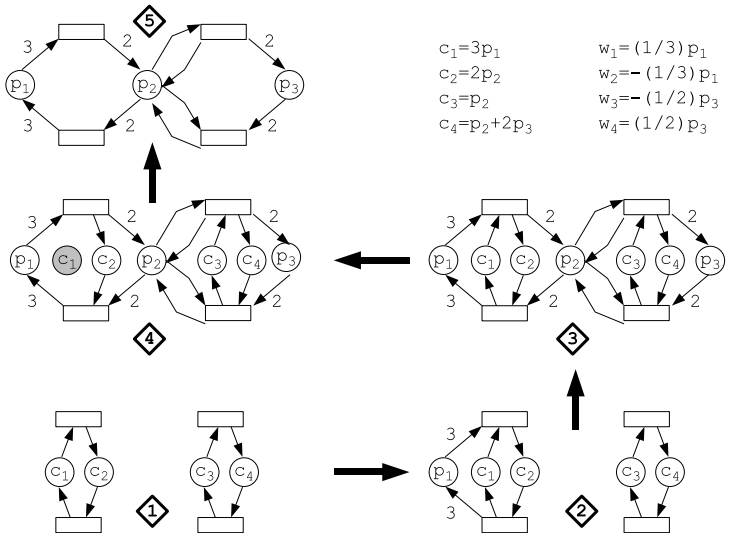


Fig. 2. How to synthetise a Π^2-net

For modelling purposes, we could define more general rules like the refinement of a place by a strongly connected state machine. Here the goal was to design a minimal set of rules.

3.2 From Non Π^2-Nets to Π^2-Nets

Below we propose a procedure which takes as input any Petri net and returns a Π^2-net. The important disclaimer is that the resulting net, although related to the original one, has a different structural and timed behaviour. So it is up to the modeller to decide if the resulting net satisfies the desired specifications. In case of a positive answer, the clear gain is that all the associated Markovian Petri nets have a product form.

Consider a Petri net $\mathcal{N} = (P, T, W^-, W^+, m_0)$ with set of complexes $\mathcal{C}$. Assume that $\mathcal{N}$ is not WR. For each transition t, add a reverse transition t^- such that ${}^\bullet t^- = t^\bullet$ and $(t^-)^\bullet = {}^\bullet t$ (unless such a transition already exists). The resulting net is WR. In the Markovian Petri net, the added reverse transitions can be given very small rates, to approximate more closely the original net.

Now, to enforce deficiency 0, the idea is to compose a general Petri net with its reaction graph as in the illustration of Proposition 3.4.

Definition 3.5. *Consider a Petri net $\mathcal{N} = (P, T, W^-, W^+, m_0)$. Let $\overline{m}_0$ be an initial marking for the reaction Petri net $\mathcal{A}$. The* regulated *Petri net associated with $\mathcal{N}$ is defined as follows:*

$$\mathcal{A} \odot \mathcal{N} = \big(P \sqcup \mathcal{C}, T, \widetilde{W}^-, \widetilde{W}^+, (m_0, \overline{m}_0)\big), \qquad \widetilde{W}^- = \begin{bmatrix} W^- \\ \overline{W}^- \end{bmatrix}, \quad \widetilde{W}^+ = \begin{bmatrix} W^+ \\ \overline{W}^+ \end{bmatrix}.$$

Proposition 3.6. *The regulated Petri net $\mathcal{A} \odot \mathcal{N}$ is WR iff $\mathcal{N}$ is WR. The regulated Petri net $\mathcal{A} \odot \mathcal{N}$ has deficiency 0.*

The behaviours of the original and regulated Petri nets are different. In particular, the regulated Petri net is bounded, even if the original Petri net is unbounded. Roughly, the regulation imposes some control on the firing sequences.

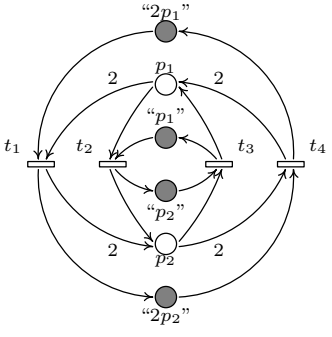


Fig. 3. Regulated Petri net associated with the Petri net of Fig 1

Consider the example of Figures 1 (original net) and 3 (regulated net). The transitions t_1 and t_4 belong to the same simple circuit in the reaction graph. Let w be an arbitrary firing sequence. The quantity $|w|_{t1} - |w|_{t4}$ is unbounded for the original net, and bounded for the regulated net.

4 Complexity Analysis of Π^2-Nets

All the nets that we build in this section are symmetric hence WR. For every depicted transition t, the reverse transition exists (sometimes implicitly) and is denoted t^-. It is well known that reachability and liveness of safe Petri nets are PSPACE-complete [5]. In [9], it is proved that reachability and liveness are PSPACE-hard for safe Π-nets and NP-hard for safe Π^2-nets. Next theorem and its corollary improve on these results by showing that the problem is not easier for safe Π^2-nets than for general safe Petri nets.

Theorem 4.1. *The reachability problem for safe Π^2-nets is* PSPACE*-complete.*

Proof. Our proof of PSPACE-hardness is based on a reduction from the QSAT problem [17]. QSAT consists in deciding whether the following formula is true

$$\varphi \equiv \forall x_n \exists y_n \forall x_{n-1} \exists y_{n-1} \dots \forall x_1 \exists y_1 \psi$$

where ψ is a propositional formula over $\{x_1, y_1 \dots, x_n, y_n\}$ in conjunctive normal form with at most three literals per clause.

Observe that in order to check the truth of φ, one must check the truth of ψ w.r.t. the 2^n interpretations of $x_1, \dots, x_n$ while the corresponding interpretation of any y_i must only depend of the interpretation of $\{x_n, \dots, x_i\}$.

Counters modelling. First we design a Π^2-net $\mathcal{N}_{cnt}$ that "counts" from 0 to $2^k - 1$. This net is defined by:

- $P = \{p_0, \dots, p_{k-1}, q_0, \dots, q_{k-1}\}$;
- $T = \{t_0, \dots, t_{k-1}\}$;
- For every $0 \leq i < k$, ${}^\bullet t_i = p_i + \sum_{j<i} q_j$ and $t_i^\bullet = q_i + \sum_{j<i} p_j$;
- For every $0 \leq i < k$, $m_0(p_i) = 1$ and $m_0(q_i) = 0$.

Observe that for every reachable marking m and every index i, we have $m(p_i) + m(q_i) = 1$. Therefore m can be coded by the binary word $\omega = \omega_{k-1} \dots \omega_0$ in which $\omega_i = m(q_i)$. The word ω is interpreted as the binary expansion of an

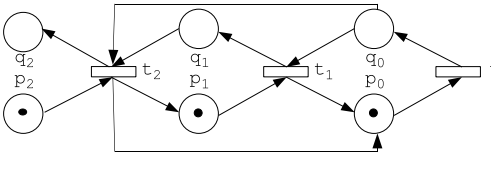
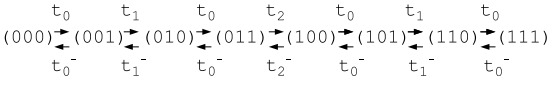


Fig. 4. A 3-bit counter (without the reverse transitions)

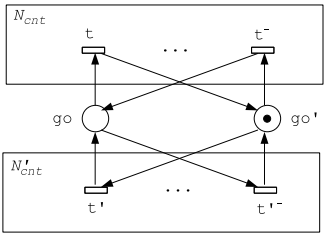
integer between 0 and $2^k - 1$. We denote by $val(\omega)$ the integer value associated with w. Consider $w \notin \{0^k, 1^k\}$, there are two markings reachable from w which are $w+$ and $w-$ such that $val(w-) = val(w) - 1$ and $val(w+) = val(w) + 1$.

The figure below represents the reachability graph of the 3-bit counter. For a k-bit counter, the shortest firing sequence from 0^k to 1^k is σ_k defined inductively by: $\sigma_1 = t_0$ and $\sigma_{i+1} = \sigma_i t_i \sigma_i$.



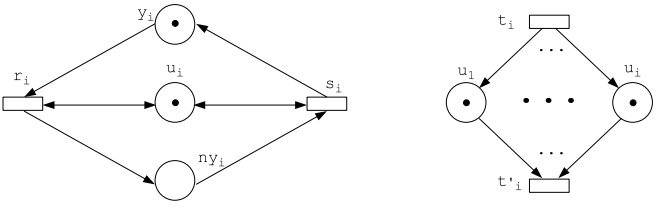
For every complex $c \equiv p_i + \sum_{j<i} q_j$ (resp. $c \equiv q_i + \sum_{j<i} p_j$), a possible witness is $wit(c) \equiv p_i + \sum_{j>i} 2^{j-i-1} p_j$ (resp. $wit(c) \equiv q_i + \sum_{j>i} 2^{j-i-1} q_j$). Thus this subnet has deficiency 0.

To manage transition firings between the update of counters, we duplicate the counter subnet and we synchronize the two subnets as indicated in the figure below. For a duplicated k-bit counter, the shortest firing sequence from the marking with the two counters set to 0^k and place go marked to the marking with the two counters set to 1^k and place go marked is obtained by: $\overline{\sigma}_1 = \overline{t}_0$ and $\overline{\sigma}_{n+1} = \overline{\sigma}_n \overline{t}_n \overline{\sigma}_n$ where $\overline{t}_i = t_i t'_i$.



This net has still deficiency 0 since the complexes are just enlarged by the places go or go' and their witnesses remain the same.

Variable modelling. For reasons that will become clear later on, the two counter subnets contain $n + 3$ bits indexed from 0 to $n + 2$. The bits $1, \ldots, n$ of counter cnt correspond to the value of variables $x_1, \ldots, x_n$. Managing the value of variables $y_1, \ldots, y_n$ is done as follows. For every variable y_i, we add the subnet described below on the left (observe that $s_i = r_i^{-1}$) and modify the two counter subnets as described on the right.



When place y_i (resp. ny_i) is marked, this corresponds to interpreting variable y_i as **true** (resp. **false**). Changes of the interpretation are possible when place u_i is marked. This is the role of the modification done on the counter subnet: between a firing of t_i and t'_i places $\{u_j\}_{j\leq i}$ are marked. With this construction, we get the expected behaviour: the interpretation of a variable y_i can only be modified when the interpretation of a variable x_j with $j \geq i$ is modified. The complexes of the counter subnet are enlarged with places u_i and their witnesses remain the same since places in the support of these witnesses are not modified by transitions s_i and r_i. The new complex $y_i + u_i$ (resp. $ny_i + u_i$) has for witness y_i (resp. ny_i). Thus the new net has still deficiency 0.

Modelling the checking of the propositional formula. We now describe the subnet associated with the checking of propositional formula $\psi \equiv \bigwedge_{j\leq m} C_j$ where we assume w.l.o.g.: (1) that every clause $C_j \equiv l_{j,1} \vee l_{j,2} \vee l_{j,3}$ has exactly three literals (i.e. variables or negated variables); and (2) that every variable or negated variable occurs at least in one clause. The left upper part of Figure 5 shows the Petri net which describes clause C_j of the formula ψ. Places $\ell_{j,k} (k = 1, 2, 3)$ represent the literals while places $n\ell_{j,k}$ represent the literal *used as a proof of the clause*, the place $mutex_j$ avoids to choose several proofs of the clause (and thus ensuring safeness), and finally place $success_j$ can be marked if and only if the evaluation of the clause yields true for the current interpretation and one of its true literal is used as a proof.

The complexes of this subnet are $mutex_j + \ell_{j,k}$ (resp. $success_j + n\ell_{j,k}$) with witness $-n\ell_{j,k}$ (resp. $n\ell_{j,k}$). So the subnet has deficiency 0.

We now synchronise the clause subnets with the previous subnet in order to obtain the final net. Observe that in the previous subnet, transition t_0 (and t'_0) must occur after every interpretation change. This is in fact the role of bit 0 of the counter. Thus we constrain its firing by requiring the places $success_j$ to be marked as presented in the right upper part of Figure 5. Adding loops simply enlarges the complexes associated with t_0 and does not modify the incidence matrix. So the net has still deficiency 0.

It remains to synchronise the value of the variables and the values of the literals where the variables occur either positively or negatively. This is done in two steps. First $\ell_{j,k}$ is initially marked if the interpretation of the initial marking satisfies $\ell_{j,k}$. Then we synchronize the value changes as illustrated in the lower part of Figure 5. Once again the complexes are enlarged and the witnesses are still valid since the places $\ell_{j,k}$ do not belong to the support of any witness.

Choice of the initial and final marking for the net. Let us develop a bit the sequence $\overline{\sigma}_{n+3}$ in the two counter subnet in order to explain the choice of initial marking for this subnet: $\overline{\sigma}_{n+3} = \overline{\sigma}_{n+1} t_{n+1} t'_{n+1} \overline{\sigma}_{n+1} t_{n+2} t'_{n+2} \overline{\sigma}_{n+1} t_{n+1} t'_{n+1} \overline{\sigma}_{n+1}$

We want to check all the interpretations of x_i's guessing the appropriate values of y_i's (if they exist). We have already seen that changing from one interpretation to another one (i.e. a counter incrementation or decrementation) allows to perform the allowed updates of y_i. However given the initial interpretation of the x_i's we need to make an initial guess of all the y_i's. So our initial marking

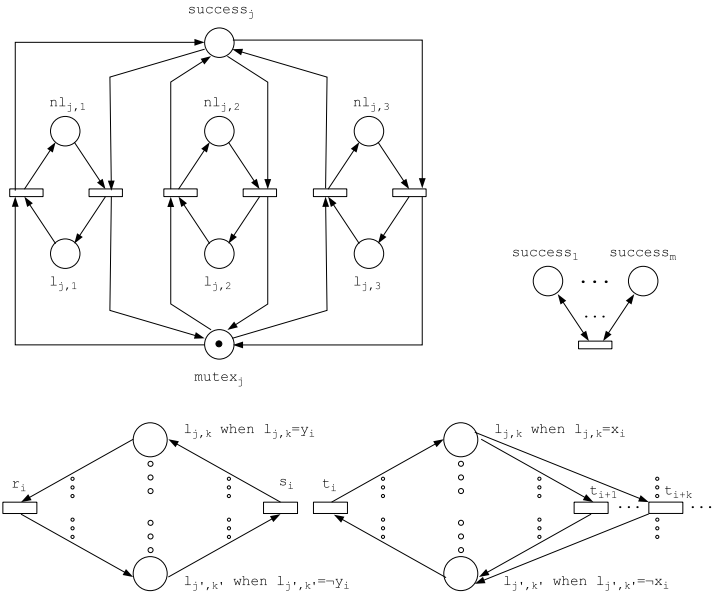


Fig. 5. Clause C_j (left), synchronisation with t_0 (right) and with variables (below)

restricted to the counter subnet will correspond to the marking reached after $\overline{\sigma}_{n+1}t_{n+1}$, i.e. corresponding to $cnt = 2^{n+1}$ (i.e. word $010\ldots0$), $cnt' = 2^{n+1} - 1$ (i.e. word $001\ldots1$) with in addition places go', u_i's, $mutex_j$'s and y_i's 1-marked; places $\ell_{j,k}$ are marked according to the initial marking of places x_i's and y_i's as explained before. All the other places are unmarked. This explains the role of bit $n+1$.

Furthermore, if we have successfully checked all the interpretations of the x_i's, the counters will have reached the value $2^{n+2}-1$ (corresponding to a firing sequence obtained from $t'_{n+1}\overline{\sigma}_{n+1}$ with possible updates of y_i during change of interpretations). However we do not know what is the final guess for the y_i's. So firing transition t_{n+2} allows to set the y_i's in such a way that the final marking will correspond to $cnt = 2^{n+2}$ (i.e. word $10\ldots0$), $cnt' = 2^{n+2} - 1$ (i.e. word $01\ldots1$) with in addition places go', u_i's $mutex_j$'s and y_i's 1-marked; places $\ell_{j,k}$ are marked accordingly. All the other places are unmarked. This explains the role of bit $n+2$.

By construction, the net reaches the final marking iff the formula is satisfied. Observe that the checking of clauses can be partially done concurrently with the change of interpretation. However as long as, in the net, a clause C_j is "certified" by a literal $\ell_{j,k}$ (i.e. marking place $success_j$ and unmarking place $\ell_{j,k}$) the value of the variable associated with the literal cannot change, ensuring that when t_0 is fired, the marking of any place $success_j$ corresponds to the evaluation of clause C_j with the current interpretation. □

Corollary 4.1. *The liveness problem for safe* Π^2*-nets is* PSPACE*-complete.*

Proof. Observe that the transitions of the net of the previous proof are fireable at least once (and so live by weak reversibility) iff φ is true. □

Let us now consider general (non-safe) Petri nets. Reachability and coverability of symmetric nets is EXPSPACE-complete [16]. In [9], it is proved that both problems are EXPSPACE-complete for WR nets (which include symmetric Petri nets). Next proposition establishes the same result for the coverability of Π^2-nets.

Proposition 4.1. *The coverability problem for Π^2-nets is* EXPSPACE*-complete.*

The complexity of reachability for Π^2-nets remains an open issue (indeed the proof of EXPSPACE-hardness does not work for reachability).

5 The Subclass of Π^3-Nets

In this section, we introduce Π^3-nets, a subclass of product-form Petri nets for which the normalising constant can be efficiently computed. The first subsection defines the subclass; the second one studies its structural properties and the third one is devoted to the computation of the normalising constant.

5.1 Definition and Properties

Definition 5.1 (Ordered Π-net). *Consider an integer $n \geq 2$. An n-level* ordered Π-net *is a Π-net $\mathcal{N} = (P, T, W^-, W^+)$ such that:*

1. $P = \bigsqcup_{1 \leq i \leq n} P_i$, $T = \bigsqcup_{1 \leq i \leq n} T_i$ *and $P_i \neq \emptyset$ for all $1 \leq i \leq n$,*
2. $\mathcal{M}_i = (P_i, T_i, W^-_{|P_i \times T_i}, W^+_{|P_i \times T_i})$ *is a strongly connected state machine,*
3. $\forall 1 \leq i \leq n\,, \forall t \in T_i\,, \forall p \in P$, ${}^\bullet t(p) > 0$ *implies $p \in P_i$ or $p \in P_{i-1}$ ($P_0 = \emptyset$),*
4. $\forall 2 \leq i \leq n\,, \exists t \in T_i\,, \exists p \in P_{i-1}$ *s.t.* ${}^\bullet t(p) > 0$,
5. $\forall 1 \leq i \leq n\,, \forall t, t' \in T_i$, $({}^\bullet t \cap {}^\bullet t') \cap P_i \neq \emptyset$ *implies* ${}^\bullet t = {}^\bullet t'$.

We call $\mathcal{M}_i$ the level i state machine. The elements of P_i (resp. T_i) are level i places (resp. transitions). The complexes ${}^\bullet t$ with $t \in T_i$ are level i complexes.

By weak reversibility, the constraints 3, 4, and 5 also apply to the output bags $t^\bullet$. An *ordered Π-net* is a sequence of strongly connected state machines. Connections can only be made between a level i transition and a level $(i-1)$ place (points 1, 2, 3). By construction, an *ordered Π-net* is connected (point 4). Each level i place belongs to one and only one level i complex (point 5).

Lemma 5.2. *The reaction net of $\mathcal{N}$ is isomorphic to the disjoint union of state machines $\mathcal{M}_i$. So a T-semi-flow of $\mathcal{M}_i$ is also a T-semi-flow of $\mathcal{N}$. If a transition of T_i is enabled by a reachable marking then every transition of T_i is live.*

An ordered Π-net may be interpreted as a multi-level system. The transitions represent jobs or events while the tokens in the places represent resources or constraints. A level i job requires resources from level $(i-1)$ and relocates these

resources upon completion. Conversely, events occurring in level $(i-1)$ may make some resources unavailable, hence interrupting activities in level i. The dependency of an activity on the next level is measured by *potentials*, defined as follows.

Definition 5.3 (Interface, potential). *A place $p \in P_i$, $1 \leq i \leq n-1$, is an* interface place *if $p \in t^\bullet$ for some $t \in T_{i+1}$. For a place $p \in P_i$, $2 \leq i \leq n$, and a place $q \in P_{i-1}$, set:*

$$pot(p,q) = \begin{cases} t^\bullet(q) & \text{if } p \text{ and } q \text{ have a common input transition } t \in T_i \\ 0 & \text{otherwise.} \end{cases}$$

The potential *of a place $p \in P_i$, $2 \leq i$, is defined by: $pot(p) = \sum_{q \in P_{i-1}} pot(p,q)$. By convention, $pot(p) = 0$ for all $p \in P_1$.*

By the definition of ordered Π-nets, the quantity $t^\bullet(q)$ does not depend on the choice of t, so the potential is well-defined.

Example. The Petri net in Figure 6 is a 3-level ordered Π-net. The potentials are written in parentheses. To keep the figure readable, the arcs between the place p_1 and the level 2 transitions are omitted.

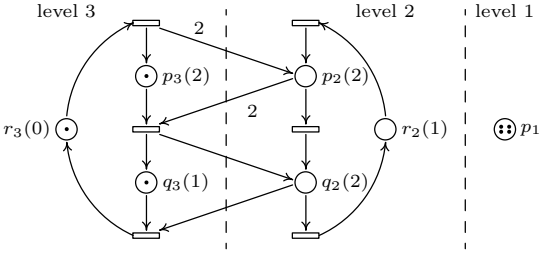


Fig. 6. Ordered Π-net

The behaviour of the state machines $\mathcal{M}_i$ is embedded in the behaviour of $\mathcal{N}$, in the sense that the marking stripped off of the potentials evolves like a marking of the state machines.

Definition 5.4 (Effective marking). *The* effective marking *of a marking m, denoted by $\widetilde{m}$, is defined as follows. For all $i \leq n$ and $p \in P_i$,*

$$\widetilde{m}(p) = m(p) + \sum_{j=1}^{n-i} \big((-1)^j \sum_{\substack{r_1 \in P_{i+1} \\ \dots \\ r_j \in P_{i+j}}} m(r_j) \big(\prod_{k=1}^{j-1} pot(r_{k+1}, r_k) \big) pot(r_1, p) \big) . \qquad (5.1)$$

Remark. Note that an effective marking is not necessarily non-negative. It can be showed by induction that:

$\forall p \in P_n , \widetilde{m}(p) = m(p)$ and $\forall p \in P_i , i < n , \widetilde{m}(p) = m(p) - \sum_{r \in P_{i+1}} \widetilde{m}(r) pot(r,p)$

Lemma 5.5. *Let m, m' be two vectors such that $m' = m + W(t)$ for some $t \in T_i$ $(1 \leq i \leq n)$. Let p_1 and p_2 denote the input place and the output place of t in P_i, respectively. Then for every place p:*

$$\widetilde{m}'(p) = \widetilde{m}(p) - 1 \text{ if } p \text{ is } p_1, \quad \widetilde{m}(p) + 1 \text{ if } p \text{ is } p_2, \quad \widetilde{m}(p) \text{ otherwise.} \qquad (5.2)$$

The above lemma applies in particular when m and m' are markings such that $m \xrightarrow{t} m'$. Eqns (5.2) look like the equations for witnesses. Since each level i complex contains exactly one level i place, one guesses that every complex admits a witness, i.e. that $\mathcal{N}$ is a Π^2-net. This is confirmed by the next proposition.

Proposition 5.6. *Let B denote the $P \times P$ integer matrix of the linear transformation $m \mapsto \widetilde{m}$ defined by (5.1). For $p \in P_i$, the line vector $B(p)$ is a witness for the i-level complex containing p. In particular, $\mathcal{N}$ is a Π^2-net.*

Lemma 5.5 allows to derive the S-invariants of $\mathcal{N}$ induced by S-semi-flows.

Corollary 5.7. *Let m_0 be the initial marking of $\mathcal{N}$. We have:*

$$\forall m \in \mathcal{R}(m_0), \quad \forall i \in \{1, \ldots, n\}, \qquad \widetilde{m}(P_i) = \widetilde{m}_0(P_i)$$

More generally, for all i, the vector $v_i = \sum_{p \in P_i} B(p)$ is a S-semi-flow of $\mathcal{N}$.

Example. Consider the ordered Π-net in Figure 6 with the initial marking $m_0 = p_3 + q_3 + r_3 + 4p_1$. The effective marking of m_0 is $\widetilde{m_0} = p_3 + q_3 + r_3 - 2p_2 - q_2 + 10p_1$. Any reachable marking m satisfies the invariants:

$$m(P_3) = 3$$
$$m(P_2) - 2m(p_3) - m(q_3) = -3$$
$$m(p_1) - 2m(p_2) - 2m(q_2) - m(r_2) + 4m(p_3) + 2m(q_3) = 10$$

It can be shown that $\{v_i, 1 \leq i \leq n\}$ is a basis of the S-semi-flows of $\mathcal{N}$.

Proposition 5.8. *Let v be an S-semi-flow of $\mathcal{N}$, i.e. $v.W = 0$. There exist rational numbers $a_1, \ldots, a_n$ such that $v = \sum_{i=1}^{n} a_i v_i$.*

The independence of the set $\{v_i, 1 \leq i \leq n\}$ follows from the fact that the vectors $v_i B^{-1}$ have non-empty disjoint supports.

We now consider only ordered Π-nets in which the interface places in P_i have maximal potential among the places of P_i. From the technical point of view, this assumption is crucial for the reachability set ananlysis presented later. From the modelling point of view, it is a reasonable restriction. Consider the multi-level model, the assumption means that during the executions of level i jobs, the level $(i-1)$ is idle, therefore the amount of available resource is maximal.

Definition 5.9 (Π^3-net). *An ordered Π-net $\mathcal{N}$ is a Π^3-net if:*

$$\forall i, \forall p \in P_i \; : p \in {}^\bullet T_{i+1} \implies pot(p) = max\{pot(q), q \in P_i\}\,.$$

5.2 The Reachability Set

From now on, $\mathcal{N}$ is a n-level Π^3-net with $\mathcal{M}_1, \ldots, \mathcal{M}_n$ its state machines.

Definition 5.10 (Minimal marked potential). *Consider $i \in \{2, \ldots, n\}$. The* level i minimal potential marked by m *is:*

$$\varphi_i(m) = \begin{cases} max\{pot(p), p \in P_i\} & \text{if } m(P_i) = 0\,, \\ min\{pot(p), p \in P_i, m(p) > 0\} & \text{if } m(P_i) > 0\,. \end{cases}$$

Next lemma gives a necessary condition for reachability.

Lemma 5.11. *If $\varphi_i(m) \leq m(P_{i-1})$ then $\varphi_i(m') \leq m'(P_{i-1})$ for all $m' \in \mathcal{R}(m)$.*

For our purposes, we now define the partial liveness and partial reachability.

Definition 5.12 (i-reachability set, i-liveness). *Let m be a marking. The i-reachability set of m, denoted by $\mathcal{R}_i(m)$, is the set of all markings reachable from m by a firing sequence consisting of transitions in $\bigcup_{1 \leq j \leq i} T_j$. We say that m is* i-live *if for any transitions t in $\bigcup_{1 \leq j \leq i} T_j$, there exists a marking in $\mathcal{R}_i(m)$ which enables t. By convention, $\mathcal{R}_0(m) = \{m\}$ and every marking is 0-live.*

The i-live markings are characterised by the following proposition.

Proposition 5.13. *A marking m is i-live if and only if it satisfies the following inequalities, called the* i-condition*:*

$$m(P_i) > 0 \wedge \forall 2 \leq j \leq i \ : \ m(P_{j-1}) \geq \varphi_j(m) \tag{5.3}$$

If m satisfies the i-condition then for every $p, q \in P_i$ such that $p \neq q$, $m(p) > 0$ and $pot(p) \leq m(P_{i-1})$, there exists $m' \in \mathcal{R}_i(m)$ such that:

$$m'(p) = m(p) - 1\,,\ m'(q) = m(q) + 1\,,\ \forall r \in P_i \setminus \{p, q\},\ m'(r) = m(r)\,. \tag{5.4}$$

A marking is live if and only if it satisfies the n-condition.

Example: The ordered Π-net in Figure 6 is a Π^3-net. Consider two markings: $m_1 = p_3 + q_3 + r_3 + 4p_1$ and $m_2 = 3q_3 + 4p_1$. These markings agree on all the S-invariants, but only m_1 satisfies the 3-condition. It is easy to check that m_1 is live while m_2 is dead.

We conclude this subsection by showing that the reachability problem for Π^3-nets can be efficiently decided as well.

Theorem 5.14. *Suppose that the initial marking m_0 is live. Then the reachability set $\mathcal{R}(m_0)$ coincides with the set $\mathcal{S}(m_0)$ of markings which satisfy the n-condition and agree with m_0 on the S-invariants given by Corollary 5.7.*

5.3 Computing the Normalising Constant

The normalising constant of a product-form Petri net (see Section 2.1) is $G = \sum_m \ _{m \in \mathcal{R}(m_0)} \prod_{p \in P} u_p^{m(p)}$. It is in general a difficult task to compute G, as can

be guessed from the complexity of the reachability problem. However, efficient algorithms may exist for nets with a well-structured reachability set. Such algorithms were known for Jackson networks [18] and the *S-invariant reachable* Petri nets defined in [6]. We show that is is also the case for the class of live Π^3-nets which is strictly larger than the class of Jackson networks (which correspond to 1-level ordered nets) and is not included in the class of S-invariant reachable Petri nets.

Suppose that m_0 is a live marking. Suppose that the places of each level are ordered by increasing potential: $P_i = \{p_{i1}, \ldots, p_{ik_i}\}$ such that $\forall 1 \leq j < k_i$, $pot(p_{ij}) \leq pot(p_{i(j+1)})$.

Let V denote the $n \times P$-matrix the i-th row of which is the S-invariant v_i defined in Corollary 5.7. For $1 \leq i \leq n$, set $C_i = v_i m_0 = \widetilde{m}_0(P_i)$. Then the reachability set consists of all n-live markings m such that $Vm = {}^t(C_1, \ldots, C_n)$.

For $1 \leq i \leq n$, $1 \leq j \leq k_i$ and $c_1, \ldots, c_i \in$, define $E(i, j, c_1, \ldots, c_i)$ as the set of markings m such that

$$\begin{cases} m(p_{i\nu}) = 0 \text{ for all } \nu > j \\ Vm = {}^t(c_1, \ldots, c_i, 0 \ldots, 0) \\ \varphi_\nu(m) \leq m(P_{\nu-1}) \text{ for all } 2 \leq \nu \leq i\,. \end{cases}$$

The elements of $E(i, j, c_1, \ldots, c_i)$ are the markings which satisfy the second part of the i-condition and the S-invariants constraints $(c_1, \ldots, c_i, 0, \ldots, 0)$ and concentrate tokens in $P_1, \ldots, P_{i-1}$ and $\{p_{i1}, \ldots, p_{ij}\}$.

With each $E(i, j, c_1, \ldots, c_i)$ associate

$$G(i, j, c_1, \ldots, c_i) = \pi(E(i, j, c_1, \ldots, c_i)) = \sum \prod_{p \in P} u_p^{m(p)}$$

the sum being taken over all $m \in E(i, j, c_1, \ldots, c_i)$.

We propose to compute $G(n, k_n, C_1, \ldots, C_n)$ by dynamic programming. It consists in breaking each $G(i, j, c_1, \ldots, c_i)$ into smaller sums. This corresponds to a partition of the elements of $E(i, j, c_1, \ldots, c_i)$ by the number of tokens in p_{ij}.

Proposition 5.15. *Let be given $E = E(i, j, c_1, \ldots, c_i)$. If $c_i < 0$ then $E = \emptyset$. If $c_i \geq 0$ then for every non-negative integer a:*

1. *If $a > c_i$ then $E \cap \{m | m(p_{ij}) = a\} = \emptyset$.*
2. *If $a < c_i$ and $j = 1$ then $E \cap \{m | m(p_{ij}) = a\} = \emptyset$.*
3. *If $a < c_i$ and $j \geq 2$ then $E \cap \{m | m(p_{ij}) = a\} = E(i, j-1, c_1 - v_1(ap_{ij}), \ldots, c_i - v_i(ap_{ij}))$:*
4. *If $a = c_i$ and $i = 1$ then $E \cap \{m | m(p_{ij}) = a\} = \{c_1 p_{1j}\}$.*
5. *If $a = c_i$ and $i > 1$ then $E \cap \{m | m(p_{ij}) = a\} = E(i - 1, k_{i-1}, c_1 - v_1(ap_{ij}), \ldots, c_{i-1} - v_{i-1}(ap_{ij}))$:*

Proposition 5.15 induces the following relations between the sums $G(i, j, c_1, \ldots, c_i)$.

Corollary 5.16. *If $c_i < 0$ then $G(i, j, c_1, \ldots, c_i) = 0$. If $c_i \geq 0$ then:*

– *Case* $2 \leq i \leq n$, $2 \leq j \leq k_i$*:*

$$G(i,j,c_1,\ldots,c_i) = \sum_{\nu=0}^{c_i-1} u_{p_{ij}}^{\nu} G(i,j-1,c_1-v_1(\nu p_{ij}),\ldots,c_i-v_i(\nu p_{ij})) + u_{p_{ij}}^{c_i} G(i-1,k_{i-1},c_1-v_1(c_i p_{ij}),\ldots,c_{i-1}-v_{i-1}(c_i p_{ij}))\,.$$

– *Case* $2 \leq i \leq n$, $j = 1$*:*

$$G(i,1,c_1,\ldots,c_i) = u_{p_{i1}}^{c_i} G(i-1,k_{i-1},c_1-v_1(c_i p_{i1}),\ldots,c_{i-1}-v_{i-1}(c_i p_{i1}))\,.$$

– *Case* $i = 1$, $j \geq 2$*:* $G(1,j,c_1) = \sum_{\nu=0}^{c_1-1} u_{p_{1j}}^{\nu} G(1,j-1,c_1-\nu) + u_{p_{1j}}^{c_1}$.
– *Case* $i = 1$, $j = 1$*:* $G(1,1,c_1) = u_{p_{11}}^{c_1}$.

Complexity. Since $i \leq n$, $j \leq K = \max\{k_1,\ldots,k_n\}$, the number of evaluations is bounded by $n \times K \times \gamma$, where γ upper bounds the c_i's. Let α denote the global maximal potential. From (5.1), we obtain $\gamma = \mathcal{O}(m_0(P)K^n\alpha^n)$. So the complexity of a dynamic programming algorithm using Cor. 5.16 is $\mathcal{O}(m_0(P)nK^{n+1}\alpha^n)$, i.e. pseudo-polynomial for a fixed number of state machines.

6 Perspectives

This work has several perspectives. First, we are interested in extending and applying our rules for a modular modelling of complex product-form Petri nets. Then we want to validate the formalism of Π^3-nets showing that it allows to express standard patterns of distributed systems. Finally we conjecture that reachability is EXPSPACE-complete for Π^2-nets and we want to establish it.

Acknowledgements. We would like to thank the anonymous referess whose numerous and pertinent suggestions have been helpful in preparing the final version of the paper.

References

[1] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Chichester (1995)
[2] Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity. John Wiley & Sons, New York (1992)
[3] Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.: Open, closed and mixed networks of queues with different classes of customers. Journal of the ACM 22(2), 248–260 (1975)
[4] Boucherie, R.J., Sereno, M.: On closed support T-invariants and traffic equations. Journal of Applied Probability (35), 473–481 (1998)
[5] Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Journal of Informatik Processing and Cybernetics 30(3), 143–160 (1994)
[6] Coleman, J.L., Henderson, W., Taylor, P.G.: Product form equilibrium distributions and a convolution algorithm for stochastic Petri nets. Performance Evaluation 26(3), 159–180 (1996)

[7] Esparza, J.: Reduction and Synthesis of Live and Bounded Free Choice Petri Nets. Information and Computation 114(1), 50–87 (1994)
[8] Feinberg, M.: Lectures on chemical reaction networks. Given at the Math. Research Center, Univ. Wisconsin (1979), http://www.che.eng.ohio-state.edu/~feinberg/LecturesOnReactionNetworks
[9] Haddad, S., Moreaux, P., Sereno, M., Silva, M.: Product-form and stochastic Petri nets: a structural approach. Performance Evaluation 59, 313–336 (2005)
[10] Henderson, W., Lucic, D., Taylor, P.G.: A net level performance analysis of stochastic Petri nets. Journal of Australian Mathematical Soc. Ser. B 31, 176–187 (1989)
[11] Jackson, J.R.: Jobshop-like Queueing Systems. Management Science 10(1), 131–142 (1963)
[12] Kelly, F.: Reversibility and Stochastic Networks. Wiley, New-York (1979)
[13] Lazar, A.A., Robertazzi, T.G.: Markovian Petri Net Protocols with Product Form Solution. In: Proc. of INFOCOM 1987, San Francisco, CA, USA, pp. 1054–1062 (1987)
[14] Li, M., Georganas, N.D.: Parametric Analysis of Stochastic Petri Nets. In: Fifth International Conference on Modelling and Tools for Computer Performance Evaluation, Torino, Italy (1991)
[15] Mairesse, J., Nguyen, H.-T.: Deficiency zero petri nets and product form. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 103–122. Springer, Heidelberg (2009)
[16] Mayr, E., Meyer, A.: The complexity of the word problem for commutative semigroups an polynomial ideals. Advances in Math. 46, 305–329 (1982)
[17] Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
[18] Reiser, M., Lavenberg, S.S.: Mean Value Analysis of Closed Multichain Queueing Networks. Journal of the ACM 27(2), 313–322 (1980)

A Tool for Automated Test Code Generation from High-Level Petri Nets

Dianxiang Xu

National Center for the Protection of the Financial Infrastructure,
Dakota State University
Madison, SD 57042, USA
dianxiang.xu@dsu.edu

Abstract. Automated software testing has gained much attention because it is expected to improve testing productivity and reduce testing cost. Automated generation and execution of tests, however, are still very limited. This paper presents a tool, ISTA (Integration and System Test Automation), for automated test generation and execution by using high-level Petri nets as finite state test models. ISTA has several unique features. It allows executable test code to be generated automatically from a MID (Model-Implementation Description) specification - including a high-level Petri net as the test model and a mapping from the Petri net elements to implementation constructs. The test code can be executed immediately against the system under test. It supports a variety of languages of test code, including Java, C/C++, C#, VB, and html/Selenium IDE (for web applications). It also supports automated test generation for various coverage criteria of Petri nets. ISTA is useful not only for function testing but also for security testing by using Petri nets as threat models. It has been applied to several industry-strength systems.

Keywords: Petri nets, predicate/transition nets, software testing, model-based testing, security testing.

1 Introduction

Software testing is an important means for quality assurance of software. It aims at finding bugs by executing a program. As software testing is labor intensive and expensive, it is highly desirable to automate or partially automate testing process. To this end, model-based testing (MBT) has recently gained much attention. MBT uses behavior models of a system under test (SUT) for generating and executing test cases. Finite state machines [9] and UML models [5] are among the most popular modeling formalisms for MBT. However, existing MBT research cannot fully automate test generation or execution for two reasons. First, tests generated from a model are often incomplete because the actual parameters are not determined. For example, when a test model is represented by a state machine or sequence diagram with constraints (e.g., preconditions and postconditions), it is hard to automatically determine the actual parameters of test sequences so that all constrains along each test sequences are satisfied [9]. Second, tests generated from a model are not immediately executable

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 308–317, 2011.

because modeling and programming use different languages. Automated execution of these tests often requires implementation-specific test drivers or adapters.

To tackle these problems, this paper presents ISTA (Integration and System Test Automation)[1], a tool for automated generation of executable test code. It uses high-level Petri nets for specifying test models so that complete tests can be generated automatically. It also provides a language for mapping the elements in test models (Petri nets) to implementation constructs. This makes it possible to convert the model-based tests into code that can be executed against the SUT.

ISTA can reduce a lot of testing workload by supporting various testing activities, such as the following:

- Functional testing: ISTA can generate functional tests to exercise the interactions among system components.
- Security testing: ISTA can generate security tests to exercise potential insecure behaviors.
- Regression testing: Regression testing is conducted when system requirements or implementation are changed. If test cases are not completely generated, tester needs to determine whether they have become invalid and whether they have to be changed. Using ISTA, however, only needs to change the specification for test generation.

The remainder of this paper is organized as follows. Section 2 introduces ISTA's architecture, its input language MID (Model-Implementation Description), and test generation features. Section 3 reports several real world applications of ISTA. Section 4 reviews and compares related work. Section 5 concludes this paper.

2 Automated Test Code Generation in ISTA

2.1 Architecture

Figure 1 shows the architecture of ISTA. The input to ISTA is a MID specification, consisting of a Predicate/Transition (PrT) net, a MIM (model-implementation-mapping), and HC (helper code). Section 2.2 will elaborate on MID.

The main components of ISTA are as follows:

- MID editor: a graphic user interface for editing a MID specification. A MID file can also be edited directly through MS Excel.
- MID parsers: The PrT net parser (2.1 in Figure 1) is used to parse the textual representation of the PrT net in a MID specification. The MIM parser (2.2 in Figure 1) is used to parse the MIM specification and helper code. The parsers check to see if there are any errors in the given specification.
- Reachability analyzer: it verifies whether or not a given goal marking can be reached from the initial marking in the PrT net of a given MID specification. The reachability analysis can help detect problems of the specification. For example, if a goal marking is known to be reachable (or unreachable) from the initial marking but the reachability analysis reports a different result, then

[1] Interested readers may contact the author for an evaluation version.

the PrT net models is not specified correctly (e.g., a precondition is missing). Reachability verification is conducted through the fast planning graph analysis adapted for the particular PrT nets in ISTA [8].

- Test generator: it generates model-level tests (i.e., firing sequences) according to a chosen coverage criterion. The tests are organized and visualized as a transition tree, allowing the user to debug MID specification and manage the tests before generating executable test code. ISTA supports a number of coverage criteria for test generation from PrT nets, including reachability graph coverage, transition coverage, state coverage, depth coverage, and goal coverage (to be discussed in Section 2.3).
- Test manager: it allows the user to manage the generated tests, such as export tests to a file, import tests from a file, delete tests, change the ordering of tests, redefine actual parameters of tests, and insert extra code into tests.
- Test code generator: it generates test code in the chosen target language from a given transition tree. Currently, ISTA supports the following languages/test frameworks: Java/JUnit (or Jfcunit), C/C++, C#/NUnit,VB, and html/Selenium IDE. JUnit is a unit test framework for Java programs. Jfcunit is an extension to JUnit for GUI testing of Java programs. NUnit, similar to JUnit, is a unit test framework for C# programs. Selenium IDE (a Fireforx plugin) is a test framework for web applications and thus ISTA is applicable to all web applications that can be tested with Selenium IDE, no matter what language is used to implement the web application (e.g., php or perl).

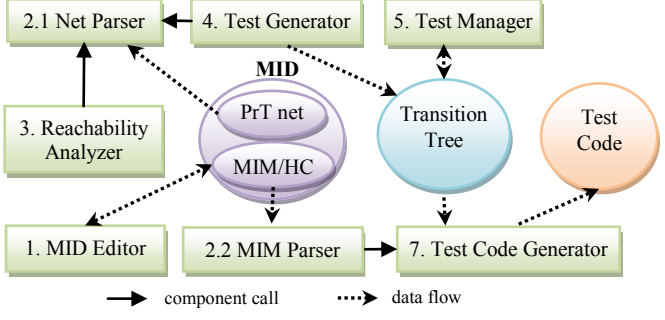


Fig. 1. The architecture of ISTA

For a complete MID specification, executable test code can be generated by just one click; it can be executed immediately against the SUT.

2.2 MID

PrT Nets as Test Models. PrT nets in ISTA are a subset of traditional PrT nets [2] or colored Petri nets [3][2] , where all weights in each formal sum of tokens or arc labels

[2] ISTA provides an interface to import CPN files. So CPN (http://cpntools.org/) can be used as a graphical editor for the PrT nets. Fig. 7 shows an example.

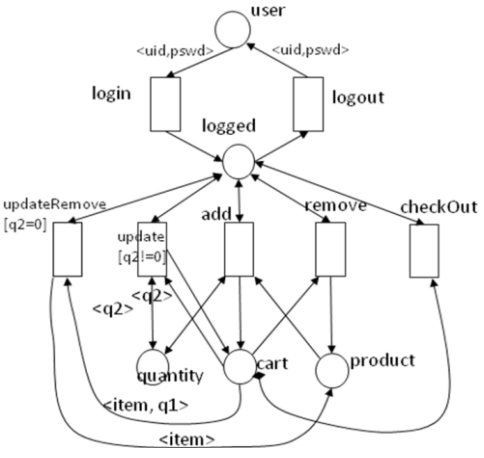


Fig. 2. Petri net for function testing of Magento

are 1. The formal definition of such PrT nets can be found in our prior work [7],[8]. The structure of such a PrT net can be represented by a set of transitions, where each transition is a quadruple <event, precondition, postcondition, guard>. The precondition, postcondition, and guard are first-order logic formulas. The precondition and postcondition are corresponding to the input and output places of the transition, respectively. This forms the basis of textual descriptions of PrT nets in ISTA. Fig. 2 shows a function test model for Magento (an online shopping application to be discussed in Section 3.2). The functions to be tested include log in, log out, add items to the shopping cart, remove items from the shopping cart, update the shopping cart, and check out. For clarity, most of the arc labels are omitted. For example, the arcs between login and logged and between logged and add are actually labeled by <uid, pswd>, respectively. Fig.3 presents its textual representation[3].

ISTA supports inhibitor arcs and reset arcs. An inhibitor arc represents a negative precondition of the associated transition. A reset arc represents a reset postcondition of the associated transition – it removes all tokens from the associated place after the transition is fired. In an online shopping system, for example, checkout empties the shopping cart no matter how many items were in the cart.

A PrT net for a function test model usually focuses on the normal behaviors of the system components. PrT nets can also be used to model security threats, which are potential attacks against a SUT. To do so, we introduce a special class of transitions, called attack transitions [7]. They are similar to other transitions except that their names start with "attack". When a PrT net is a threat model, we are primarily interested in the firing sequences that end with the firing of an attack transition. Such a firing sequence is called an attack path, indicating a particular way to attack a SUT.

[3] As indicated in Fig. 3, ISTA allows a test model to be specified as a set of contracts (preconditions and postconditions in first-order logic) or a finite state machine. Nevertheless, PrT nets are a unified representation of test models in ISTA. It automatically transforms the given contracts or finite state machine into a PrT net.

Model | MIM | Helper Code

Model Type: ◉ Petri net ○ Contract ○ Finite state machine

No	Transition	Precondition	Postcondition	When
1	login(uid, pswd)	user(uid, pswd)	logged(uid, pswd)	
2	logout(uid, pswd)	logged(uid, pswd)	user(uid, pswd)	
3	addItem(item, q)	logged(uid, pswd), product(item), quantity(q)	logged(uid, pswd), cart(item, q)	not equals(q, 0)
4	removeItem(item, q)	logged(uid, pswd), cart(item, q)	logged(uid, pswd), product(item)	
5	updateItem(item, q1, q2)	logged(uid, pswd), cart(item, q1), quantity(q2)	logged(uid, pswd), cart(item, q2), quantity(q2)	not equals(q2, 0)
6	updateRemove(item, q1)	logged(uid, pswd), cart(item, q1), quantity(q2)	logged(uid, pswd), product(item), quantity(q2)	equals(q2, 0)
7	checkout(uid, pswd)	logged(uid, pswd), cart(item, q)	logged(uid, pswd), reset(cart)	

Initial State [1]

user(UID, PSWD),

Fig. 3. Textual representation of the PrT net in Fig. 2

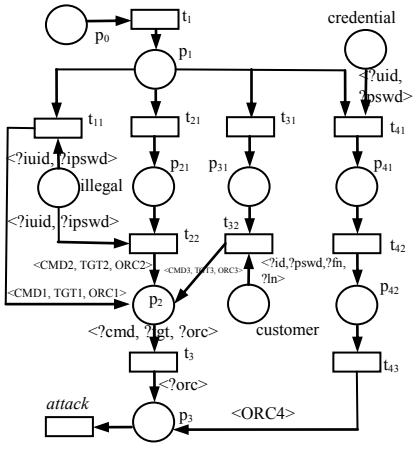


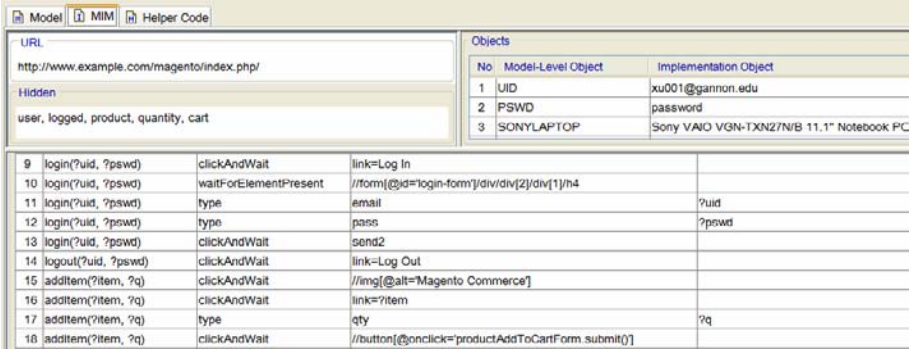
Fig. 4. Petri net for a threat model of Magento

We have demonstrated that PrT nets can be used to capture various security threats [7], such as spoofing, tampering with data, repudiation, information disclosure, denial of service, and elevation of privilege.

Fig. 4 shows a PrT net that models a group of XSS (Cross Site Scripting) threats, which are one of the top security risks in web applications. The threat model captures several ways to exploit system functions by entering a script into an input field, such as email address, password, or coupon code. The functions are log in (t1, t11, t3), create account (t1, t21, t22, t3), forgot password (t1, t31, t32, t3), and shopping with discount coupon (t1, t41, t42, t43, t3). Using formal threat models for security testing can better meet the need of security testing to consider "the presence of an intelligent adversary bent on breaking the system" [10].

MIM (Model-Implementation Mapping). A MIM specification for a test model mainly consists of the following elements:

- The identity of the SUT (URL of a web application, class name for an object-oriented program, or system name for a C program).
- A list of hidden predicates in the test model that do not produce test code (e.g., test code may not be needed for a predicate that represents an ordering constraint in the test model).
- A mapping from the objects (elements in tokens) in the test model to implementation objects (object mapping).
- A mapping from events/transitions in the test model to implementation code (method mapping).
- A mapping from the predicates (places) in the test model to implementation code for verifying the predicates in the SUT (called accessor mapping for testing object-oriented programs).
- A mapping from the predicates in the test model to implementation code for making the predicates true (e.g., for setting up an option or preference) in the SUT (called mutator mapping for testing object-oriented programs).


Model | MIM | Helper Code

URL
http://www.example.com/magento/index.php/

Hidden
user, logged, product, quantity, cart

Objects

No	Model-Level Object	Implementation Object
1	UID	xu001@gannon.edu
2	PSWD	password
3	SONYLAPTOP	Sony VAIO VGN-TXN27N/B 11.1" Notebook PC

9	login(?uid, ?pswd)	clickAndWait	link=Log In	
10	login(?uid, ?pswd)	waitForElementPresent	//form[@id='login-form']/div/div[2]/div[1]/h4	
11	login(?uid, ?pswd)	type	email	?uid
12	login(?uid, ?pswd)	type	pass	?pswd
13	login(?uid, ?pswd)	clickAndWait	send2	
14	logout(?uid, ?pswd)	clickAndWait	link=Log Out	
15	addItem(?item, ?q)	clickAndWait	//img[@alt='Magento Commerce']	
16	addItem(?item, ?q)	clickAndWait	link=?item	
17	addItem(?item, ?q)	type	qty	?q
18	addItem(?item, ?q)	clickAndWait	//button[@onclick='productAddToCartForm.submit()']	

Fig. 5. Portion of a MIM specification

Fig. 5 shows portion of the MIM for the test model in Fig. 2. UID and PSWD are two objects in the test model, representing user id and password. When they appear in a test case, they refer to xu001@gannon.edu and password in the SUT, respectively. Rows 9-18 are part of the method mapping. Rows 9-13 are Selenium IDE commands for login, and row 14 is the Selenium IDE command for logout. These commands can be easily obtained by using the record function of Selenium IDE.

Helper Code. Helper code refers to the code that needs to be provided by tester in order to make the generated test code executable. It includes the header (e.g., package/import statements in Java and #include statements in C), alpha/omega segments (invoked at the beginning/end of a test suite), setup/teardown methods (invoked at the beginning/end of each test case), and extra code to be inserted into the generated test code (called local code).

2.3 Test Generation

Test generation consists of two components: test sequence generation and test code generation. Test sequence generation produces a test suite, i.e., a set of test sequences

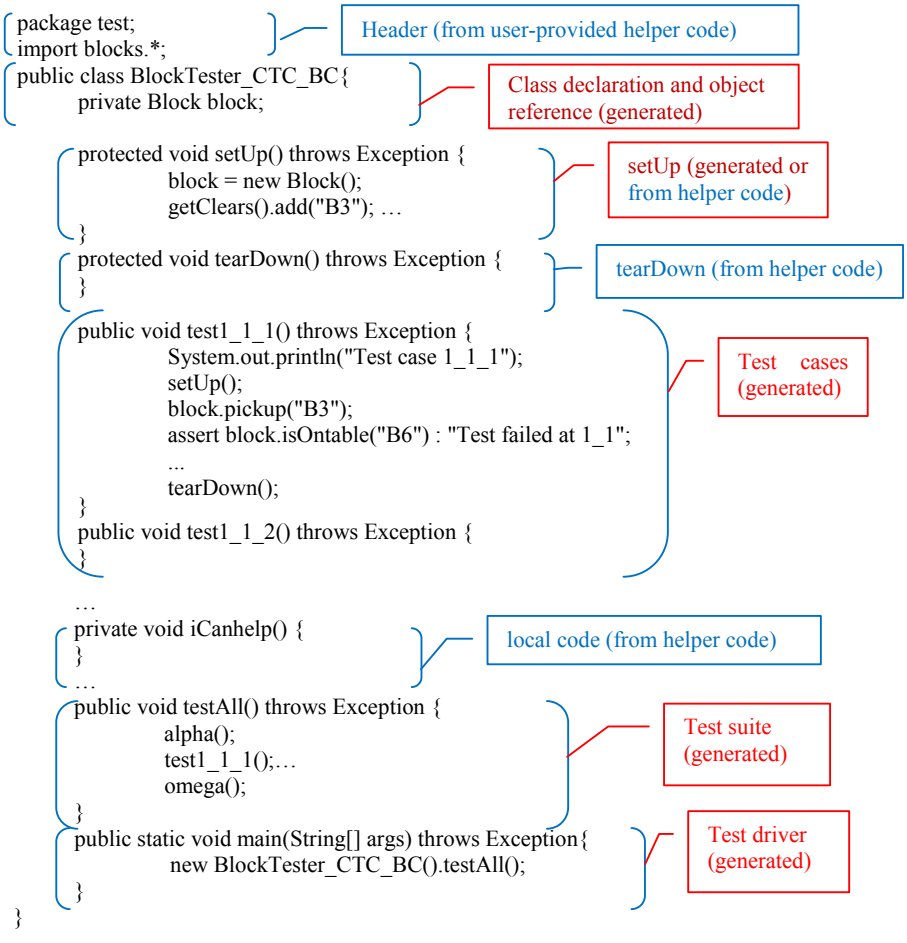


Fig. 6. Structure of the test code in Java

(firing sequences) from a test model according to a chosen coverage criterion. The test sequences are organized as a transition tree (also called test tree). The root represents the initial state (resulting from the new operation, like object construction in an object-oriented language). Each path from the root to a leaf is a firing sequence.

Given a finite state test model, ISTA can generate a transition tree to meet the following criteria:

- Reachability graph coverage: the transition tree actually represents the reachability graph of the PrT net for a function test model. If the PrT net is a threat model, however, the transition tree consists of all attack paths, i.e., firing sequences that end with the firing of an attack transition.
- Transition coverage: each transition in the PrT net is covered by at least one firing sequence in the transition tree.

- State coverage: each state (marking) reachable from the initial marking is covered by at least one firing sequence in the transition tree.
- Depth coverage: the transition tree consists of all firing sequences whose size (number of transition firings) is no greater than the given depth.
- Goal coverage: the transition tree consists of a firing sequence for each of the goal markings reachable from the initial marking.

Test code generation is to convert a transition tree to test code according to the MIM specification and helper code. The entire tree represents a test suite and each firing sequence from the root to a leaf is a test case. Fig. 6 shows the general structure of Java test code. It is similar for other object-oriented languages(C#, C++, and VB). The structure for a procedural (C) or scripting language (html) is much simpler.

3 Applications

3.1 Self-Testing of ISTA

ISTA is implemented in Java. It has been developed in an incremental fashion - new functions and examples were added from time to time. The current version has 35K LOC (lines of code) in 195 classes. Every time ISTA was updated, we had to run all the examples to check if the changes have inadvertently affected the existing code. Such regression testing would be very tedious if it were not automated. We created a test model for ISTA itself so that it can generate executable test code to run all the examples with various system options (e.g., coverage criteria). Fig. 7 shows the test model edited with CPN, where for clarity the initial marking is simplified. The automated self-testing not only saved us a lot of time, but also helped us find various problems in the development process.

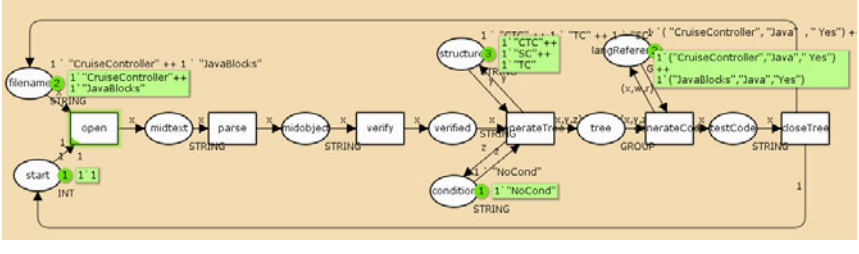


Fig. 7. PrT net for the function testing of ISTA

3.2 Function and Security Testing of Online Shopping Systems

ISTA has been applied to two real-world online shopping systems, Magento[4] and Zen-cart[5]. They are both implemented in php. Magento has 52K LOC whereas Zen-cart has 96K LOC. They both are being used by many live online stores. For

[4] www.magentocommerce.com
[5] www.zen-cart.com

Magento, more than 2,100 function tests with a total of 608K line of html code were generated and executed. For Zen-cart, more than 20,000 function tests (12 million lines of html code) were generated and executed. We have also built 19 PrT nets for various security threats to Magento. They produced 103 security tests that killed 56 of the 62 security mutants of Magento created according to OWASP's top 10 security risks of web applications and the security requirements of online shopping. A security mutant is a variant of the original version with a vulnerability injected deliberately. A test is said to kill a security mutant if it is a successful attack against the mutant.

3.3 Security Testing of an FTP Server Implementation

FTP (File Transfer Protocol) is a widely used method for working with remote computer systems and moving files between systems. FileZilla Server[6]is a popular FTP server implementation. Currently, it is the seventh most downloaded program on SourceForge[7]. FileZilla Server version 0.9.34 used in our study has 88K lines of C++ code in 107 classes. We created 8 PrT nets to specify various security threats to FileZilla Server. 76 security tests were generated. They killed 35 of the 38 security mutants of FileZilla Server created according to the security requirements of FTP services and the common vulnerabilities of C++ programs.

4 Related Work

ISTA is related to testing with Petri nets and MBT. Zhu and He [11] have proposed a methodology for testing high-level Petri nets. The methodology consists of four testing strategies: transition-oriented testing, state-oriented testing, data flow-oriented testing, and specification-oriented testing. Each strategy is defined in terms of an adequacy criterion for selecting test cases and an observation scheme for observing a system's dynamic behavior during test execution. It is not concerned with how tests can be generated to meet the adequacy criteria. Desel et al. [1] have proposed a technique to generate test inputs (initial markings) for the simulation of high-level Petri nets. Wang et al. [6] have proposed class Petri net machines for specifying inter-method interactions within a class and generating method sequences for unit testing. Manual work is needed to make the sequences executable.

There is a large body of literature in MBT, particularly with finite state machines and UML models. Existing MBT tools are limited in the generation of executable tests. We refer the reader to [9] for a brief survey of MBT research. To the best of our knowledge, ISTA is a unique tool for generating executable test code in various languages from the specifications of complex SUTs. It is the only tool that can generate executable security tests from formal threat models. These features are made possible because of the expressiveness of high-level Petri nets for capturing both control and data flows of complex software.

[6] http://filezilla-project.org/
[7] http://sourceforge.net/top/topalltime.php?type=downloads

5 Conclusions

We have presented the ISTA tool for the automated generation of executable tests by using high-level Petri nets to build the test models. Using ISTA for its own testing has proven to be an effective approach in its incremental development process. As demonstrated by several case studies, ISTA is applicable to both function testing and security testing of various software systems. It can improve testing productivity and reduce testing cost due to automated generation and execution of tests.

ISTA has been adopted by a globally leading company in high-tech electronics manufacturing and digital media. We are currently collaborating on integrating ISTA with their testing process and extending ISTA to generate test code for concurrent programs. With the advances in multi-core technologies, concurrent software is becoming more and more ubiquitous (e.g., in cell phones and home appliances). As an excellent formalism for modeling concurrent systems, Petri nets are expected to play an important role in quality assurance of concurrent software.

Acknowledgments. This work was supported in part by the National Science Foundation of USA under grant CNS 0855106. Dr. Weifeng Xu, Michael Sanford, Lijo Thomas, Daniel Woodraska, and Jayanth Gade contributed to the application, experimentation, and documentation of ISTA.

References

1. Desel, J., Oberweis, A., Zimmer, T., Zimmermann, G.: Validation of Information System Models: Petri Nets and Test Case Generation. In: Proc. of SMC 1997, pp. 3401–3406 (1997)
2. Genrich, H.J.: Predicate/Transition Nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 207–247. Springer, Heidelberg (1987)
3. 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, 213–254 (2007)
4. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proc. of the IEEE 77(4), 541–580 (1989)
5. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, San Francisco (2006)
6. Wang, C.C., Pai, W.C., Chiang, D.-J.: Using Petri Net Model Approach to Object-Oriented Class Testing. In: Proc. of SMC 1999, pp. 824–828 (1999)
7. Xu, D., Nygard, K.E.: Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets. IEEE Trans. on Software Engineering. 32(4), 265–278 (2006)
8. Xu, D., Volz, R.A., Ioerger, T.R., Yen, J.: Modeling and Analyzing Multi-Agent Behaviors Using Predicate/Transition Nets. International Journal of Software Engineering and Knowledge Engineering 13(1), 103–124 (2003)
9. Xu, D., Xu, W., Wong, W.E.: Automated Test Code Generation from Class State Models. International J. of Software Engineering and Knowledge Engineering 19(4), 599–623 (2009)
10. Xu, D.: Software Security. In: Wah, B.W. (ed.) Wiley Encyclopedia of Computer Science and Engineering, vol. 5, pp. 2703–2716. John Wiley & Sons, Inc, Hoboken (2009)
11. Zhu, H., He, X.: A Methodology for Testing High-Level Petri Nets. Information and Software Technology 44, 473–489 (2002)

The ePNK: An Extensible Petri Net Tool for PNML

Ekkart Kindler

Informatics and Mathematical Modelling, Technical University of Denmark
eki@imm.dtu.dk

Abstract. The *Petri Net Markup Language* (*PNML*) is an XML-based interchange format for all kinds of Petri nets, which is an ISO/IEC International Standard 15909-2 now. The focus of this standard is on PNML as an interchange format for high-level Petri nets. PNML, however, is more general and allows exchanging all kinds of Petri nets. To this end, PNML introduced the concept of *Petri Net Type Definitions*.

There are many tools supporting one form of PNML or another. In particular, there is the *PNML Framework*, which helps tool developers implementing an interface to PNML by providing a framework and an API for loading and saving Petri net documents in PNML. This framework is based on the *Eclipse Modeling Framework* and has the focus on the underlying meta-models of Petri nets. The PNML Framework, however, is not *generic* in the following sense: Whenever a new Petri net type is created, the code for the complete tool needs to be regenerated. Moreover, the PNML Framework does not come with a graphical editor.

The ePNK overcomes these limitations: It provides an extension-point so that new Petri net types can be plugged into the ePNK without touching the code of the ePNK. For defining a new Petri net type, the developer, basically, needs to give a class diagram defining the concepts of the new Petri net type, along with a mapping of these concepts to XML syntax. This type can then be plugged into the ePNK, and its graphical editor will be able to edit nets of this new type with all its features. This paper present the main idea of the ePNK, and how to use and extend it.

Keywords: Extensibility, Petri net tool, Petri Net Markup Language (PNML), ISO/IEC 15909.

1 Introduction

The *ePNK* is a generic and extensible tool for all kinds of Petri nets which is based on the *Petri Net Markup Language* (*PNML*). It is implemented based on Eclipse and the *Eclipse Modeling Framework* (*EMF*) [1]. For motivating the ideas of the *ePNK*, we will explain PNML in a nutshell in this section, and in the end give an overview of the ePNK and its objective in Sect. 1.3.

1.1 Motivation

The PNML [2,3,4] is an XML-based interchange format for all kinds of Petri nets, that allows different tools to exchange Petri net models among each other. One of

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 318–327, 2011.

its main features is that it is generic, which means that it provides a mechanism for defining own types of Petri nets, called *Petri Net Type Definitions* (*PNTD*): they define the additional concepts of the new Petri net type, as well as the representation of these concepts in XML-syntax.

Though there are many tools supporting some form of PNML, there is no tool yet that fully supports all of the Petri net types of ISO/IEC 15909-2, comes with a graphical editor, allows easily plugging in new Petri net types, and can serve as a tool development platform. The ePNK fills this gap.

1.2 PNML: An Overview

In order to understand the concepts of the ePNK, we give a brief overview of the main ideas and concepts of the PNML. For more information on PNML and ISO/IEC 15909-2, we refer to [4,5,6].

As stated above, the extensibility and genericity were two of the main objectives behind the PNML [2]. This is achieved by identifying the concepts that are common to all Petri nets in the so-called *PNML core model*. These common concepts are mainly the *places*, *transitions* and *arcs*, and that these objects can have some kind of *labels*. The PNML core model also takes into account that larger Petri nets can be split up into *pages* and connections between the nodes on the different pages can be established by *reference places* and *reference transitions*. And there are all kinds of graphical information that can be attached to the different net elements. In addition, the PNML core model defines the relations between these elements. In particular, it defines that places and transitions, which are generalized as *nodes*, are contained in pages and that arcs may connect these nodes. Figure 1 shows the main concepts of the PNML core model as a UML class diagram. Note that there is only one concrete type of label in the PNML core model, which is *name*. All the other possible labels need to be defined by a Petri net type definition, which will be discussed later.

In addition to the PNML concepts and their relations, the PNML core model also states some restrictions. For example, there is a constraint stating that arcs can only be between nodes that are on the same page, which is formulated in the *Object Constraint Language* (*OCL*). But, there is no constraint in the PNML core model, which states that arcs can run only from a place to a transition or the other way round. The reason for that is that there are some kinds of Petri nets that would allow arcs between places or between transitions. Therefore, these restrictions are part of the respective Petri net type definition.

Note also that the PNML core model does not specify concrete tool specific extensions (ToolInfo). It is up to a tool to define what it needs. But, any tool must be able to read – and later write again – any tool specific extension; their contents however, can be ignored.

Petri net type definitions define which labels are possible in a specific kind of Petri net, and also some additional constraints. Here, we explain this idea by the help of a simple example: Place/Transition-Systems (P/T-Systems).

The two additional kinds of labels for P/T-Systems are the initial marking for places and the inscription for arcs. The initial marking can be any non-negative

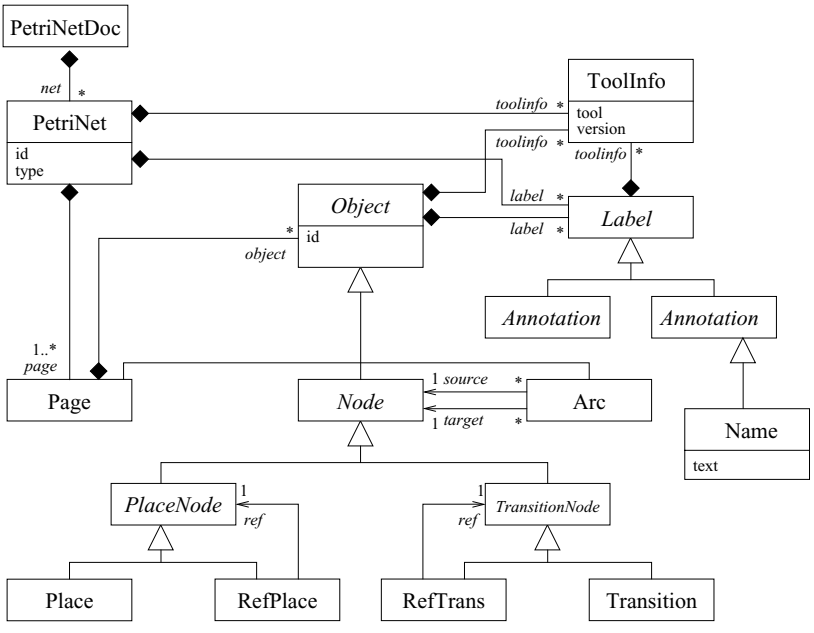


Fig. 1. The PNML core model

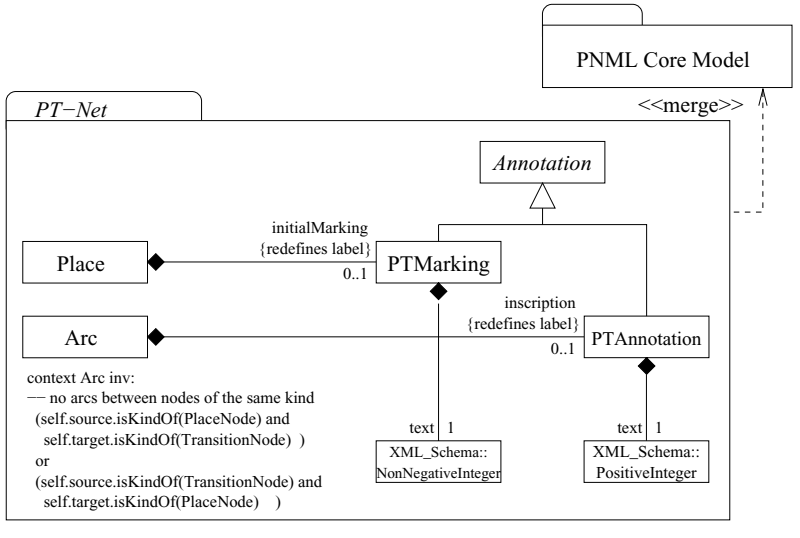


Fig. 2. The PNTD for PT-Nets

number and the inscription for arcs can be any positive number. Figure 2 shows the class diagram for these concepts and how they relate to the concepts of the PNML core model. These two models are combined by UML's merge mechanism.

Figure 2 shows an example of an OCL constraint: without going into the details of OCL, this constraint states that an arc must run from a place to a transition or from a transition to a place. So, for P/T-Systems, it is no longer legal to connect places with places or transitions with transitions.

A Petri net type definition would typically also define how the new concepts from Fig. 2 would be mapped to XML. In this simple case, however, the XML syntax can be derived from the name of the associations and the name of the attributes in the class diagram (see [4] for details).

1.3 ePNK: Objective

The ePNK is a tool and a tool platform that fully supports the concepts of PNML, so that new Petri net types along with the mapping to XML syntax can be easily plugged in. It support all the Petri net types defined in ISO/IEC 15909-2: i. e. P/T-nets, and three different versions of high-level Petri nets (HLPNGs).

As soon as a new Petri net type definition is plugged in, the ePNK is able to load and save nets of this type. And there is a graphical editor that allows us to edit Petri nets of any plugged-in net type and is fully aware of all the features and the additional restrictions of the plugged-in net types. This *end user* perspective of the ePNK will be briefly discussed in Sect. 2.

In addition to the Petri net type definitions, the ePNK provides an API to easily load and access Petri nets from PNML files, to manipulate them programmatically, and to save them. A *developer* can easily plug in new functionality for analysing and manipulating Petri nets. This way, the ePNK provides a platform for Petri net tool development and for implementing new functions. This *developers'* perspective of the ePNK is discussed in Sect. 3.

In the end, Sect. 4 provides you with the references to all the detailed information you need to download, install, and to work and develop with the ePNK.

2 ePNK: For Users

In this section, we discuss the ePNK from the end users' perspective, whom we call *users* in this paper. We explain how to create, edit, save and load PNML documents. For lack of space, we give a brief overview only. Users who want to work with the ePNK are referred to Chapt. 3 of the ePNK Manual [7].

2.1 Creating and Editing Nets

Figure 3 shows a screenshot of the Eclipse workbench with a PNML document open in the ePNK *editors* (3). On the left-hand side, the complete document with three Petri nets of different types is shown as a tree structure; on the right-hand side, a page of the first Petri net, which is a high-level net (HLPNG), is open in the graphical editor of the ePNK. At the bottom (5), you can see the *properties view*, which shows some details of the currently selected element: the type label of a place in this example. Note that the main editor for PNML documents is always the tree editor, which can be opened by double-clicking on a PNML

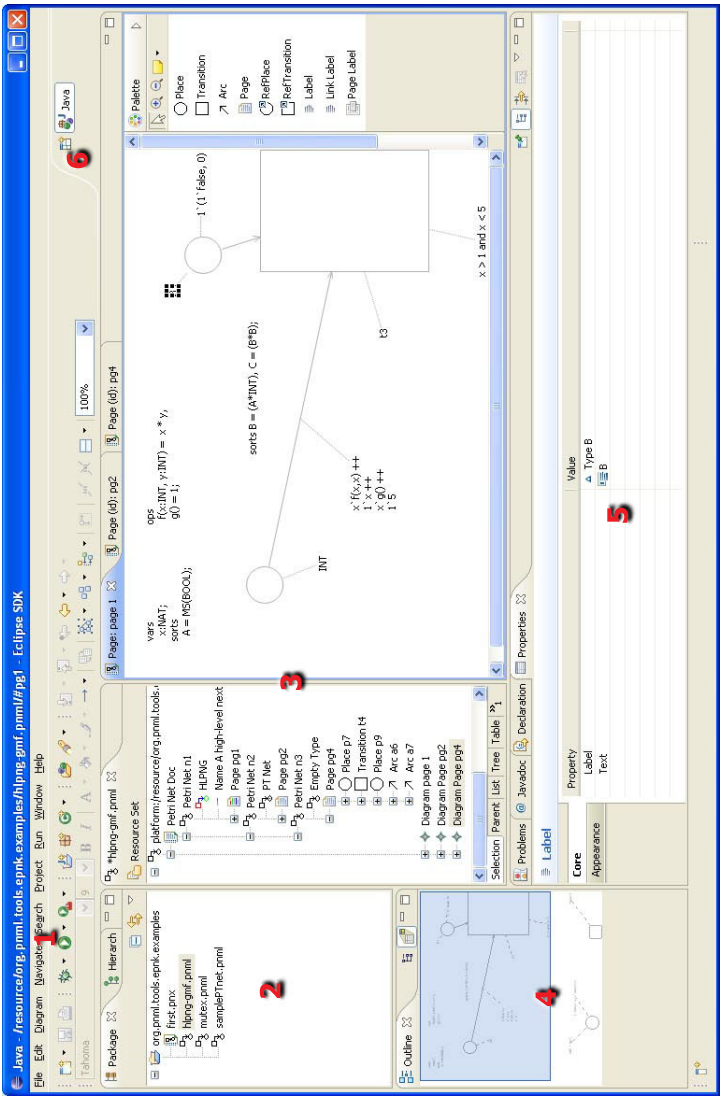


Fig. 3. The Eclipse workbench with ePNK editors open

document in the package explorer (2). The graphical editors can be opened for every page separately by right-clicking on the respective page in the tree editor and selecting "ePNK"→"Start GMF Editor on Page" in the pop-up menu.

All the elements on a page can and should be edited with the graphical editor, but saving should always be done from the tree editor. All the Petri net elements can be created via the tools in the toolbar of the graphical editor in the usual way. The only special features are *labels* and *page labels*. Since labels of PNML are specific to a Petri net type, the ePNK editor will ask the user which kind of

page label should be created by a pop-up menu, when a page label is created – the available choices take into account which labels exist already and which labels may still be created. For labels that are attached to a Petri net element this is done in two steps: first, a label is created, which will show up as a "not connected label"; second, by choosing the tool "Link Label" and dragging the mouse from a net element to that label, the label will be attached to the element, and the user can select from a pop-up menu, which kind of label it is supposed to be.

Before you can edit a page, you need to create a net and a page in it with the ePNK tree editor. And you also must add a type for this net; this works via the "Create Child" pop-up menu of the standard tree editors of the *Eclipse Modeling Framework*. For HLPNGs, you can also select a specific kind, which restricts the legal data types and operations that can be used in the nets: symmetric nets, pt-hlpng nets, and full HLPNGs.

Note that there are two different kinds of net types: *simple net types* have textual labels only, which are not interrelated and do not need particular parsing. P/T-Systems are an example for a simple net type. *Structured nets* have labels with more complex structure and have relations between different labels: e. g. a variable defined in a declaration can be used in arc inscriptions or transition conditions. These links must be established at some point: to this end, right-click on the respective Petri net and select "ePNK"→"Link labels (globally)". This will check and establich all the references between the labels "behind the scenes".

Most objects within a PNML document must have a unique identifier. In order not to bother the user with adding the IDs manually, there is a pop-up action on the Petri net document that adds all missing IDs automatically: right-click on the respective document and select "ePNK"→"Add missing IDs". The ePNK will save (and load again) Petri nets with missing IDs; but the saved file won't be conformant to PNML since the IDs are required. In order to make sure that the saved file conforms to PNML (and to the constraints of a specific type), you can select "Validate" (via a right-click on a Petri net document, a net, or some other Petri net object): a dialog will show you a list of problems; you can see the found problems later in the Eclipse "Problems" view. Actually, the concrete Petri net types and HLPNGs have many constraints, which concerns syntatical correctnes of labels, and correct typing, which are too many to be discussed here (see Sect. 3.5 of [7]).

For creating new PNML files, the ePNK comes with a wizard: it can be started via the "New" menu, which can be either accessed by the "File" menu from the menu bar or via a pop-up menu that opens on a right-mouse click in the explorer. Then, select "Other..." and, in the newly opened "Select a wizard" dialog, chose "ePNK (PNML)", and follow through the dialogs. Note that there is another creation wizard in the ePNK category, which will create a Petri net document in XMI format (which we call PNX, but is not discussed here).

2.2 Functions

The ePNK itself is not so much about providing functionality, but about enabling developers in providing functionality and help them accessing PNML documents.

For demonstrating its extension mechanisms, however, the ePNK is deployed with a simple CTL model checker for P/T-Systems, which is based on the MCiE[1] framework. With MCiE, implementing this model checker within the ePNK took just a few hours.

Once you have created a PNML document with a P/T-System, you can start the model checker by right-clicking on the net and then selecting "ePNK"→ "Model checker". In the opened dialog, you can enter a comma-separated list of CTL formulas with the usual CTL operators, where the variables refer to place names (see Sect. 4.3.2 of [7] for details). For example, you could use the automatically created PNML document multi_mutex.pnml, which is available from the ePNK site, and check the formula `AG AF semaphor`. Since model checking can take some time, the model checker is run as a background job, which will be indicated as a running bar in the Eclipse *progress area*. Once the job is finished, the running bar turns into an exclamation mark, and the result can be opened by clicking on it; then, a dialog will show for every CTL formula whether it is true or false in the Petri net.

One particular function, which is an extension of the ePNK might be quite important: Serialising the labels of a HLPNG to their ePNK syntax. This is important because ISO/IEC 15909-2 does not mandate a particular concrete syntax for labels. Their text might even be empty. This function allows you to convert the labels to ePNKs syntax so that you can edit them properly. The function is installed as a popup menu on Petri nets in the tree editor.

3 ePNK: For Developers

More interestingly, the ePNK provides mechanisms for extending it, which concerns defining new Petri net types, new tool specific extensions, and new functions. In this section, we give a brief overview of what can be extended and an idea of what needs to be done. For more detailed information on how to do it, we refer to Chapt. 4 of the ePNK Manual [7].

For defining a new Petri type, a developer needs to provide a UML class diagram that defines the labels of the new Petri net type, which very much resembles the one from Fig. 2. Technically, this model must be an ecore model (in a nutshell, ecore is the EMF version of UML class diagrams). Apart from this technical difference, there are some minor conceptual ones: since the EMF technology does not support merging packages, the classes Place and Arc in this package are actually new classes that inherit from the respective classes in the PNML core model. Moreover, the package must contain an explicit class representing the new Petri net type. Another difference is that the constraints from Fig. 2 are plugged in separately.

Basically, providing a model of the Petri net type is the only thing we need to do to plug in a simple Petri net type such as P/T-Systems. The actual code for the extension can be generated by the help of the EMF technology completely automatically: only 4 lines need to be added or change manually; then, the

[1] See `http://www2.cs.uni-paderborn.de/cs/kindler/Lehre/MCiE/`

Listing 1.1. Mapping for partition elements

```
metadata.add(
    PartitionsPackage.eINSTANCE.getPartition_PartitionElements(),
    PartitionsPackage.eINSTANCE.getPartition(),
    PartitionsPackage.eINSTANCE.getPartitionElement(),
    "partitionelement",
    null, null);
```

new type can be made known to Eclipse, which, basically, is a reference to the generated class of the Petri net type (see Sect. 4.3.1.3 of [7]). The OCL constraint that forbids connecting places to places or transitions to transitions is plugged in by the standard Eclipse mechanism (see Sect. 4.3.1.4 of [7]). For a type similar to P/T-nets, all this should not take much more than an hour.

For more complex types, it might be necessary to explicitly define the mapping from the concepts of the ecore model to the XML syntax in PNML. This applies in particular to HLPNGs. To this end, the ePNK provides a mechanism, where a list of these mappings can be defined (similar to the tables given in [4]). This table is given programmatically by calling a method for every line of the table. Programming this table is straight forward: Listing 1.1 shows the mapping of the element "partitionelement" (a construct from symmetric nets) to the XML element `<partitionelement>`: The first line refers to the reference "partitionelement" from class "Partition" to class "PartitionElement", which are explicitly mentioned in the second and third line again – in "EMF technology speak". Most of the mappings look like this, but there are some more complicated ones, which need some more sophisticated concepts. These concepts are discussed in Sect. 4.3.2 of the ePNK Manual [7]. Note that there is no need to provide a RELAX NG grammar. Tool specific extensions can be plugged into the ePNK in a way similar to simple Petri net types (see Sect. 4.4 of [7]).

The actual functionality is plugged into the ePNK with the standard Eclipse mechanisms (via actions and commands). And the implementation of these functions uses the API, that is defined by the PNML core model, and the models for the Petri net types (see Sect. 4.2 of [7]). In addition, there are some "helper methods" that help to access a Petri net with pages and reference nodes without caring about the pages and reference nodes at all (see Sect. 4.2.3 of [7]). Moreover, the Eclipse resource mechanism can be used to easily open, access, and save the contents of a document in PNML syntax or in PNX syntax (see Sect. 4.2.1 and 4.2.2 of [7]).

4 Installation and Documentation

The ePNK is implemented as an extension for Eclipse and therefore runs on all platforms that support Eclipse. To be more precise, the ePNK (as of version 0.9.1) runs under Eclipse 3.5 (Galileo) and Eclipse 3.6 (Helios).

All the information on how to obtain and install Eclipse, as well as the ePNK Manual can be found at: `http://www2.imm.dtu.dk/~eki/projects/ePNK/`. In addition, you will find some example PNML documents on these pages.

If you want to run the ePNK as a user, we would recommend at least 1GB of main memory and a not too slow hard disk (Eclipse uses many files behind the scenes). If you would like to use the ePNK as a developer, you would probably want to use a computer with 2GB of main memory.

The ePNK is fully compatible with ISO/IEC 15909-2 [5], except for the following differences:

- In the ePNK, labels can be attached to nodes and pages of a net (called Petri net objects). ISO/IEC 15909-2 also allows labels to be directly attached to a net. We call these labels *net labels*. These net labels cannot be defined with the ePNK yet, and Petri nets that have net label, cannot be read by the ePNK. The easiest way to read them is by surrounding the net labels in the file with an XML element `<page>` in some text editor.
- The ePNK allows an attribute "name" in ArbitrayOperators, which is not allowed in ISO/IEC 15909-2. If such a name is present in a Petri net document of the ePNK that is saved to a file, the resulting file is not strictly compatible with ISO/IEC 15909-2. But, the ePNK can read PNML files without the name attribute, and the ePNK will never create such a name, unless the user would explicitly create it. Therefore, this is a minor issue.
- The ePNK does not make use of all the graphical features of PNML yet. The only graphical information used in the ePNK are positions and dimensions of nodes, labels, and intermediate points of arcs. But the ePNK can read all the graphical information of PNML files (and will write them again, if they were present) and it produces PNML conformant graphical information.

5 Conclusion

The ePNK is a generic and extensible Petri net tool, which can be used for all kinds of Petri nets. PNML is its native file format, which allows it to exchange the nets with other tools. What is more, the model-based philosophy of PNML was also used as the development philosophy of the ePNK – based on the EMF technology and GMF for the graphical editors.

Major parts of the ePNK development as well as of defining new types was making the right models – and generating the rest fully automatically. Adding functions still needs some programming – using the API generated from the models. Anyway, the ePNK should help easing the implementation of prototypic Petri net tools – which is also demonstrated by the overall development time of the ePNK itself, which was a bit more than 5 weeks (up to version 0.9.0, all ISO/IEC 15909-2 Petri net types included).

This actually, was the dream we dreamt when first talking about a universal Petri net tool and the Petri Net Kernel (PNK) in the mid 90ties [8,9]. This is why we called this new tool ePNK, for *eclipse-based PNK*, though the ePNK is not compatible with the PNK.

The functionality of the ePNK goes much beyond the pure serialisation API of the PNML Framewok [10]. It provides a generic editor, allows us to plugin types, toolspecific information, and new functionality.

Up to now, the ePNK was developed by me as the only developer. I believe that by its features and by PNML as its native file format, it could become a good platform for many scientific tool developments. This way, the community that is necessary for creating a sustainable code (and model) base and with rich and reliable functionality could be established. If you want to join in, you are more than welcome.

References

1. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework, 2nd edn. The Eclipse Series. Addison-Wesley, Reading (2006)
2. Jüngel, M., Kindler, E., Weber, M.: Towards a generic interchange format for Petri nets – position paper. In: Bastide, R., Billington, J., Kindler, E., Kordon, F., Mortensen, K.H. (eds.) Meeting on XML/SGML based Interchange Formats for Petri Nets, pp. 1–5 (2000)
3. Billington, J., Christensen, S., van Hee, K.M., 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. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Treves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. In: Jensen, K. (ed.) 10^{th} Workshop on Coloured Petri Nets (CPN 2009), pp. 101–120 (2009)
5. ISO/IEC: Systems and software engineering – High-level Petri nets – Part 2: Transfer format, International Standard ISO/IEC 15909-2:2011
6. Kindler, E.: The Petri Net Markup Language and ISO/IEC 15909-2: Concepts, status, and future directions. In: Schnieder, E. (ed.) Entwurf komplexer Automatisierungssysteme, Fachtagung, vol. 9, pp. 35–55 (2006) invited paper
7. Kindler, E.: ePNK: A generic PNML tool - users' and developers' guide: version 0.9.1. IMM-Technical Report-2011-03, DTU Informatics, Denmark (2011)
8. Kindler, E., Desel, J.: Der Traum von einem universellen Petrinetz-Werkzeug — Der Petrinetz-Kern. In: Desel, J., Oberweis, A., Kindler, E. (eds.) 3. Workshop Algorithmen und Werkzeuge für Petrinetze. Forschungsberichte, Institut AIFB, Universität Karlsruhe, vol. 341 (1996)
9. Kindler, E., Weber, M.: The Petri Net Kernel – an infrastructure for building Petri net tools. Software Tools for Technology Transfer 3(4), 486–497 (2001)
10. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML framework: An extendable reference implementation of the Petri Net Markup Language. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 318–327. Springer, Heidelberg (2010)

Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models

Michael Westergaard*

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
m.westergaard@tue.nl

Abstract. This paper introduces Access/CPN 2.0, which extends Access/CPN with high-level primitives for interacting with coloured Petri net (CPN) models in Java programs. The primitives allow Java programs to monitor and interact with places and transitions during execution, and embed entire programs as subpages of CPN models or embed CPN models as parts of programs. This facilitates building environments for systematic testing of program components using a CPN models. We illustrate the use of Access/CPN 2.0 in the context of business processes by embedding a workflow system into a CPN model.

1 Introduction

Coloured Petri nets (CPNs) have proved to be a useful formalism for different modelling tasks, including verification of network protocols and modelling of business processes. CPNs are a combination of Petri nets and a programming language, Standard ML. The use of a real programming language for inscriptions rather than a specially tailored inscription language has allowed usage of CPN models beyond modelling, effectively using the CPN modelling language as an implementation language. The aim of Access/CPN 2.0 is to allow programmers to use CPN models in programs in a more high-level manner, either by embedding a CPN model as a component of a program, by embedding a program as part of a CPN model, or by observing and interacting with places and transitions of CPN models during model execution. All a programmer has to do is adhere to an interface, and Access/CPN 2.0 takes care of all synchronization. Access/CPN 2.0 supports symmetric communication between CPN models and programs without polling on either side.

An early attempt of integrating coloured Petri nets is [5], where a library for communication between CPN models and (Java) components is devised with the purpose of invoking code written in other languages and vice versa by means of a generic remote procedure call mechanism. In [6] the authors use a CPN

* This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 328–337, 2011.

model as the back-end of a web-service for simulating influence nets for operational planning using an ad-hoc means of communication, and in [4] a CPN model is used as server for a course-of-action tool for scheduling military operations using the general-purpose communication library Comms/CPN [3] to implement a remote procedure call mechanism. Common for these solutions is that they make communication explicit in the model, potentially cluttering it. From another point of view, interaction has additionally been provided with visualisation libraries, such as Mimic/CPN [10] and the successor, the BRITNeY Suite [15], which allow modellers to set up domain-specific visualisations of CPN models, letting users interact with models indirectly using domain-specific user-interfaces. Other tools, like Renew [11], allow Java inscriptions, making it directly possible to execute Java code when a transition is executed, at the cost of cluttering the model and allowing synchronous communication only. Most recently, the Access/CPN [13] library has been used to interact with CPN models using the simulator component of CPN Tools [2]. This allows scenarios where the simulation is directed by the user program or simulation results are shown to the user in a domain specific way without altering models. Unfortunately, the Access/CPN library is working at a fairly low level, such as executing a transition or getting the marking of a place, making the creation of such applications more demanding than necessary as the programmer has to handle synchronization manually. Access/CPN 2.0 aims to improve on these deficiencies by taking care of synchronization and not requiring cluttering models.

As a simple example of the use of Access/CPN 2.0 let us look at a CPN model of a simple workflow system (WS) in Fig. 1. At the top level (Fig. 1 (left)), we have two components, a Workflow System and a User, which communicate using 9 interface places. The interface is most easily described by its use, and in Fig. 1

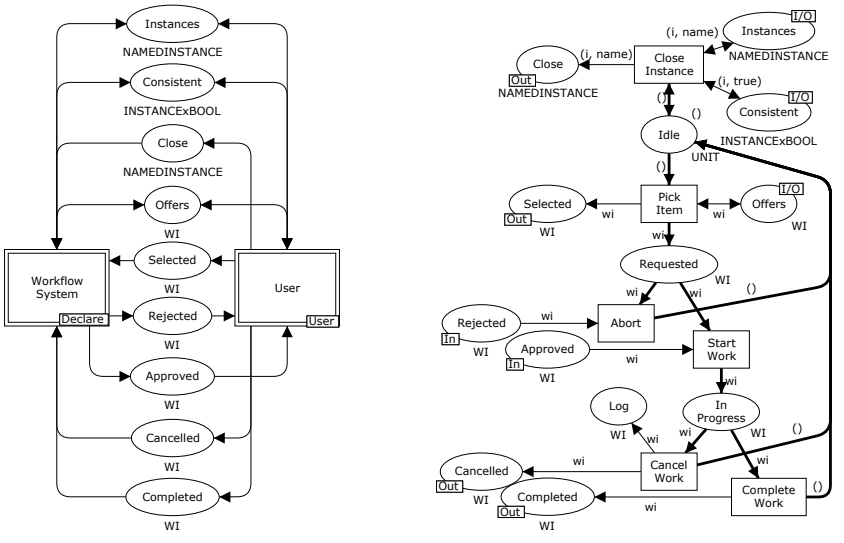


Fig. 1. CPN model of simple workflow system

(right) we see a simple CPN implementation of a User module. A user starts in the Idle state, and can choose to Pick Item. The work item (wi) is chosen from Offers, which is a read-only set of available items provided by the WS. After the user has picked an item, it is put on the Selected place to signal to the WS that it has been picked. The WS may either reject or accept a request for an item by moving the item to either Rejected or Accepted. If it is rejected, the user Aborts and returns to the Idle state, otherwise the user can Start Work and transition to the In Progress state. Here the user may either choose to Cancel Working on the item or to Finish Working, signalling the choice by moving the item to either Cancelled or Completed and returning to the Idle state. If work is cancelled, we log this so we can investigate why (by adding the item to the place Log). In the Idle state the user can additionally choose to close an instance of a workflow. Instances are provided by the read-only list Instances, and an instance can only be closed of it is Consistent (i.e., if all required items have been completed). Closing an instance is signalled by moving it to Close.

Our goal is to take the model in Fig. 1 and use it with a real workflow system; Access/CPN 2.0 reduces the effort needed to achieve such a task, and works on two levels: at the simple level, programs can interact with places and transitions during execution, and at the higher level, programs can act as submodules of CPN models or vice versa. Access/CPN 2.0 does this without requiring any annotation of CPN models, without having to focus on every detail of the simulation, and using real notification mechanism instead of polling.

Interaction between programs and places/transitions allows programs to be notified when a transition is executed or when the marking of a place is modified, similar to the monitoring facilities of CPN Tools. Access/CPN 2.0 implements these features in Java, which is more widely known than Standard ML. Access/CPN 2.0 additionally allows the programmer to modify the execution. In the example in Fig. 1, we could, e. g., monitor the Log place and store the logged data in a database for further processing, we could monitor the Start Work, Cancel Work, and Complete Work to measure efficiency of the user, or we could monitor and interact with the Pick Item transition to present a list of available work items to a user and even let the execution of the model depend on the user's choice.

Access/CPN 2.0 allows programs to embed CPN models or CPN models to embed programs. It can be useful to embed a CPN model as part of a program, e. g., if the program contains parts that are more easily described using a CPN model, or if no implementation is available yet and a stub implementation in the form of a model is used instead. It is useful to embed programs in CPN models if a CPN model is part of a larger system, where other components already exist either in other modelling languages or as real implementations. It is possible to directly use the existing component instead of recreating it as part of the model and potentially introducing errors and requiring extra effort. As a special case, it is possible to use CPN models for model-based testing of existing components: embed the component into a CPN model, model the environment of the component as a CPN model, and use the model to exercise the component.

In the example in Fig. 1, we would like to embed a real workflow system instead of having to model one. This could be useful for evaluating which of various strategies for picking work items results in the fastest execution.

The remainder of this paper is structured as follows: in the next section, we introduce the architecture of Access/CPN 2.0 and introduce the provided primitives. In Sect. 3, we show a couple of interesting applications of the primitives in the context of business process management and process mining. Finally, in Sect. 4, we conclude and provide directions for future work. Access/CPN 2.0 and video demonstrations are available from `cpntools.org/accesscpn/` and runs on all major platforms. All examples are included in ProM [9] nightly builds and installable in the upcoming ProM 6.1 release.

2 Architecture

In this section we introduce the architecture of Access/CPN 2.0 and the primitives offered. We regard Access/CPN 2.0 and Access/CPN as separate components the latter can be used without the Access/CPN 2.0 features. The architecture of Access/CPN 2.0 and required tools and libraries is shown in Fig. 2. From right to left we have the CPN Tools GUI, the CPN Tools Simulator, and an application using Access/CPN 2.0. CPN Tools is used as the editor for CPN models and can read and write `.cpn` files. These files can be read by Access/CPN, represented in memory, and sent to the CPN Tools Simulator, which generates model-specific code to simulate the model. Access/CPN is able to communicate with the generated model-specific simulator code to execute transitions, and get and set the marking of places. Access/CPN 2.0 builds on top of Access/CPN. Instead of providing a CPN model specific interface like Access/CPN, Access/CPN 2.0 aims to provide a more abstract interface supporting monitoring places and transitions, and for embedding CPN models as parts of programs or vice versa. On top of Access/CPN 2.0 we have added integration with ProM [9], which additionally adds a GUI. The dotted line between the CPN Tools GUI and CPN Tools Simulator indicates that it is possible for the two components to communicate but not necessary for operation (models can just be generated or loaded from `.cpn` files without starting the CPN Tools GUI). Similarly, the line between the application using Access/CPN 2.0 and `.cpn` files is dotted to show we can but do not have to load a `.cpn` file. the CPN Tools GUI).

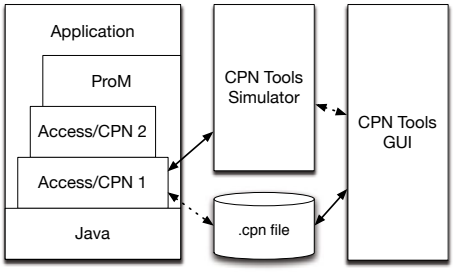


Fig. 2. Architecture of Access/CPN 2.0

The main idea of Access/CPN 2.0 is to perform a *cosimulation* between a CPN model and Java classes, i. e., simulate the CPN model and run the Java classes at the same time, synchronizing when required. We have described a more generic

cosimulation in [14], so the interface can be used by any modeling language and programming language as long as they adhere to our interfaces. The cosimulation in [14] is focused on simulation, whereas we focus on using models together with real program. We we believe that CPN models and Java is the most natural combination for this, so we have focused on making this integration simple rather than implementing a generic interface. Naturally, interaction between the CPN simulator (written in Standard ML) and Java classes imposes an overhead, but we are not too concerned with performance, as we expect the usage to mostly concentrate on integrating one or more large components in a CPN model, so communication is limited.

To control the life cycle of Java classes participating in a cosimulation, we require that they implement the interface CPNToolsPlugin from Fig. 3 (ll. 1–3), containing methods that are called before and after a cosimulation. The start method takes a single parameter, an ExecutionContext, which is a key-value mapping used by plug-ins to store data between call-backs from the cosimulation and to communicate between plug-ins. The exact use of this is not dictated by Access/CPN 2.0. A main building block of the interface of Access/CPN 2.0 is a unidirectional channel (ll. 5–6). InputChannel allows clients to receive data by calling the getOffers method and OutputChannel allows sending data using offer. All values exchanged are of type Value, which comprises all values supported

```
01 interface CPNToolsPlugin {
02   void start(ExecutionContext context);
03   void end(); }

05 interface InputChannel { Collection<Value> getOffers(); }
06 interface OutputChannel { void offer(Collection<Value> offers); }

08 interface PlacePlugin extends CPNToolsPlugin, InputChannel, OutputChannel { }

10 interface TransitionPlugin extends CPNToolsPlugin {
11   boolean isEnabled(Binding binding, Number time);
12   Binding execute(Binding binding, Number time, Collection<String> rebindable); }

14 interface DataStore {
15   Collection<Value> getValues();
16   void setValues(Collection<Value> values); }

18 interface SubpagePlugin extends CPNToolsPlugin {
19   void setInterface(Collection<InputChannel> inputs, Collection<OutputChannel> outputs,
20                     Collection<DataStore> data);
21   boolean isDone(); }

23 interface CPNSimulation {
24   Collection<InputChannel> getInputs();
25   Collection<OutputChannel> getOutputs();
26   Collection<DataStore> getData();
27   void done(); }

29 interface CosimulationManager {
30   Cosimulation setUp(PetriNet model, Map<Instance<Page>, SubpagePlugin> subpagePlugins,
31                      Map<Instance<Place>, PlacePlugin) pPlugins,
32                      Map<Instance<Transition>, TransitionPlugin> tPlugins);
33   CPNSimulation launch(Cosimulation cosimulation);
34   void launchInCPNTools(Cosimulation cosimulation); }
```

Fig. 3. Access/CPN 2.0 API

natively by CPN Tools. Access/CPN 2.0 defines three kinds of plug-ins and one interface for embedding CPN models: one plug-in for observing places (l. 8), one for observing transitions (ll. 10–12), and one to be plugged in instead of a subpage (ll. 18–21), as well as an interface representing the entire model (ll. 23–27).

A plug-in to observe places, a PlacePlugin (Fig. 3, l. 8), is a plug-in which implements the general super interface CPNToolsPlugin and the two channel interfaces. It acts as a bidirectional channel, and whenever a token is produced on the place associated with an instance of such a plug-in, it is offered to the plug-in (and removed from the model). Similarly, the plug-in can produce tokens on the associated place by offering them using getOffers. Tokens produced by a place plug-in are made available to the model as soon as the transition currently being executed (if any) is done, and tokens are removed from places and offered to place plug-ins immediately after execution of the transition producing them. Place plug-ins do not need to take explicit measures to achieve this synchronization.

A TransitionPlugin (Fig. 3, ll. 10–12), extends the common superinterface with two methods, isEnabled and execute. isEnabled is invoked before a transition is executed and can be used to reject executing a transition in a binding even though it is enabled in the underlying CPN model. execute is invoked when a transition observed by this plug-in is executed. Binding is part of Access/CPN and contains information about the transition instance executed and the binding of all variables surrounding it, providing all information needed to monitor an execution together with the time parameter, which describes when the transition was executed. To additionally allow modification of the execution, the parameter rebindable contains a collection of names of all variables that can be rebound by the plug-in. This means that by rebinding variables we can choose among different bindings of the transition. Formally, we strive for passing a set of rebindable variables such that, in the original model, if a binding b is enabled, then any binding b' with $b(v) = b'(v)$ for any v not in rebindable, is also enabled. We can approximate this statically by the set of free variables of a transition, i. e., the variables not occurring on any arc to the transition nor in the guard of the transition. Unfortunately, this rejects several variables we would like to rebind, namely any variable of a color set which is *large*, which roughly speaking means color sets containing more than 100 elements, thereby excluding integers and strings. This is a limitation of the simulator of CPN Tools, and as we would like to keep our models executable in CPN Tools without Access/CPN 2.0, we cannot work around that. Instead, we split a binding element up in a pre- and a postset, indicating tokens to remove when executing the binding and tokens to produce. We execute the preset of the binding element passed as parameter to execute and the postset of the binding element returned by the method. While altering the semantics of models, it works well in practise if we only rebind variables designed to be rebound, for example if variables are free except for being set to a value in the guard or if they are only bound using a double arc to a place. In our workflow example in Fig. 1, we would like to monitor and alter the execution of Pick Item. Here, we can safely rebind wi to any task available on Offers even though wi is not a free variable. As with place plug-ins,

transition plug-ins do not need to handle synchronization themselves; simulation is automatically paused while a transition plug-in is consulted.

A plug-in representing a subpage of a CPN model, a SubpagePlugin (Fig. 3, ll. 18–21), contains a method for defining the interface between the CPN model and the plug-in, setInterface, and a method allowing the plug-in to indicate when it is done processing, isDone. The interface consists of a collection of input parameters, a collection of output parameters, and a collection of data parameters. Input parameters correspond to input port places, i. e., the plug-in receives information from the surroundings by means of channels in this collection, so whenever a token is produced on the place in the CPN model corresponding to a parameter (a socket place), it can be received using the getOffers method of the input parameter. In the example in Fig. 1, the places Close, Selected, Cancelled, and Completed correspond to input parameters when using a subpage plugin for the Workflow System substitution transition. Output parameters play the same role for outputs and are in the example represented by the places Rejected and Approved. Data parameters are a restricted combination of input and output parameters. The plug-in is allowed to specify the values of the data store (and they correspond exactly to tokens on a corresponding socket place), but the CPN model is not allowed to modify that place. In our example, the places Instances, Consistent, and Offers correspond to data stores. Offering values on input parameters to subpage plug-ins, consumption of values produced on output parameters, and updates for datastores are guaranteed to be performed between transitions in the model. Subpage plug-ins may however need to guarantee that tokens are produced on places (corresponding to either output parameters or data stores) atomically, i.e., so no transition can be executed between the production on one place and on another. To achieve this, a subpage plug-in can synchronize on the ExecutionContext.

We do not provide an interface for plug-ins that can embed CPN models, but rather the other way around. The interface, CPNSimulation (Fig. 3, ll. 23–27) is the dual of the interface for adding subpages to a CPN model: it contains a way to get input and output parameters and data-stores of the model. The meaning of these is the same as for subpage plug-ins, except the channels now no longer correspond to socket places in the CPN model, but rather to port places on the top page. The dual of the isDone operation of the subpage plug-in is the done method indicating that the surrounding code is done with the CPN model, and Access/CPN 2.0 can stop simulating the model and free any allocated resources.

Access/CPN 2.0 provides a mechanism for setting up a cosimulation using plug-ins implementing the interfaces from Fig. 3 ll. 1–27 and executing it. This mechanism is shown in Fig. 3 ll. 29–34. setUp ties a CPN model together with place, transition and subpage plug-ins, and returns a Cosimulation, which can either be launched, resulting in a CPNSimulation which can be used by programs embedding CPN models, or the cosimulation can be launched in a special way that lets the user control the simulation directly from within the CPN Tools GUI (which is useful when embedding external code using subpage plug-ins).

3 Use Cases

Whereas the interfaces offered by Access/CPN 2.0 may seem simple, they are powerful in practise. We have implemented several CPNToolsPlugins and a ProM [9] plug-in allowing us to interactively construct a cosimualtion for execution. These examples, aside from the last one, are not intended to provide completely new functionality (for example, monitoring has previously been described in [7] and ProM orchestration in [1]), but just to give an idea of what is possible with Access/CPN 2.0, and the complexity of performing such tasks.

Showing Simulation Feedback. Sometimes we just want to run a simulation and see the end result, e. g., if the model implements a computation. For displaying results, we create a place plug-in, alerting users when a token is produced on observed places, and a transition plug-in reporting executed transitions.

Generating Execution Logs. Process mining is concerned with generating models from execution logs. One way of validating the usefulness of a process mining algorithm is to create a model, run executions of the model, and do process mining on the resulting data, comparing the result with the original model. We can easily do this using Access/CPN 2.0 by creating a transition plug-in which creates a log in the ExecutionContext and adds transition executions whenever they occur.

ProM Orchestration. ProM consists of many plug-ins, each of which consumes input values and produces output values. Normally, a user manually executes plug-ins but for complicated or repetitive tasks it is desirable to be able to automate this. A way of doing so is to use a CPN model do describe the workflow inside ProM and make sure that ProM executes plug-ins corresponding to elements of the model. We can do so using a transition plug-in and Access/CPN 2.0, identifying transitions of the CPN model with plug-ins of ProM and tokens with values. We can use mapping in the ExecutionContext and the binding of the transition to compute the actual parameters to pass to a plug-in when executing a transition, and rebind variables on out-going arcs according to the token/value mapping.

Embedding the Declare Workflow Engine and Operational Support. It would be nice to be able to use a real workflow engine directly from within a CPN model like the one in Fig. 1. We have a workflow engine called Declare, which allows us to specify workflows declaratively, and we would like to embed that in our model at it allows to focus on the task at hand, such as evaluating different strategies for picking tasks rather than modeling a workflow system. ProM supports operational support, i. e., on-line process mining aiming to aid a user with exactly that kind of decision. Operational support algorithms can be implemented in many ways. By embedding a model of an operational support algorithm, we can make it available to clients as if it was natively implemented. We can combine a plug-in embedding Declare with a plug-in embedding ProM's operational support engine into a single suite for evaluating operational support algorithms.

3.1 Summary

Table 1 summarise the plug-ins mentioned in this paper along with a classification of the type of plug-in and the complexity of the plug-in measured by lines of code (LoC) excluding boiler-plate code to illustrate that with Access/CPN 2.0 and relatively little effort, nontrivial tasks can be solved. We have provided only the LoC as a metric for effort put into the plug-ins, as most of the described plug-ins have been developed together with Access/CPN 2.0, and therefore the time spent on each individual plug-in is nontrivial to measure. One plug-in, ProMOrchestrate, was developed on its own, and took about 4 hours in one sitting to develop from scratch to completion.

Table 1. Overview of plug-ins

Name	Type	LoC
InputValue	Place	1
ShowValue	Place	1
ShowTransition	Transition	38
GenerateLog	Transition	88
ProMOrchestrate	Transition	254
Declare	Subpage	403
OSServer	Superpage	200

4 Conclusion and Future Work

We have presented Access/CPN 2.0 and applications of it, primarily within the domain of business process management, but from the diversity of the examples we believe they support the general applicability of Access/CPN 2.0.

Many previously available libraries for communication between CPN models and external code work on a relatively low level, either providing communication primitives [6, 4, 3], remote procedure call inspired interfaces of more or less generality [5, 10, 15] or detailed access to the simulator [13]. In [12] we outlined a method for declaratively tying a model to an external program by separating the control between internally executed transitions (the model) and externally (the program), but the synchronization is either complicated and specified imperatively, or limited and domain-specific. In [14] we extended this idea to allow components to be cosimulated. In this case we used CPN modules and SystemC models as components. The high-level architecture [8] has similar aims for general simulation engines. Both of these interfaces are more focused on simulation, whereas we are more interested in using CPN models as parts of actual programs.

Using a CPN model as part of an implementation is a nice way to go quickly from a specification to a prototypical implementation, but for some applications the overhead of simulating a CPN model including the communication overhead may be too large. Therefore, approaches exist to automatically or semi-automatically generate template code for an implementation of a modelled system. It would be nice to fuse the functionality of Access/CPN 2.0 with such a code generator, allowing the generated code to directly interact with the plug-ins used by the model, thereby (mostly) filling in the templates generated from the model. Future work also includes evaluating the overhead imposed by the cosimulation, e.g., in comparison with the monitoring facilities in CPN Tools.

References

1. Cabac, L., Denz, N.: Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. In: ToPNoC. LNCS, vol. 5100, pp. 86–103. Springer, Heidelberg (2008)
2. CPN Tools webpage, `cpntools.org`.
3. Gallasch, G., Kristensen, L.M.: A Communication Infrastructure for External Communication with Design/CPN. In: Proc. of Third CPN Workshop. DAIMI-PB, vol. 554, pp. 79–93 (2001)
4. Kristensen, L.M., Mechlenborg, P., Zhang, L., Mitchell, B., Gallasch, G.E.: Model-based Development of a Course of Action Scheduling Tool. STTT 10(1), 5–14 (2007)
5. Kummer, O., Moldt, D., Wienberg, F.: Symmetric Communication between Coloured Petri Net Simulations and Java-Processes. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 690–690. Springer, Heidelberg (1999)
6. Lindstrøm, B., Wagenhals, L.W.: Operational Planning using Web-Based Interfaces to a Coloured Petri Net Simulator of Influence Nets. In: Proc. of Formal Methods Applied to Defence Systems. CRPIT, vol. 12, pp. 115–124 (2002)
7. Lindstrøm, B., Wells, L.: Towards a Monitoring Framework for Discrete-Event System Simulations. In: Proc. of WODES 2002, pp. 127–134. IEEE Computer Society Press, Los Alamitos (2002)
8. Morse, K.L., Lightner, M., Little, R., Lutz, B., Scrudder, R.: Enabling Simulation Interoperability. Computer 39(1), 115–117 (2006)
9. Process mining webpage, `processmining.org`.
10. Rasmussen, J.L., Singh, M.: Mimic/CPN. A Graphical Simulation Utility for Design/CPN. User's Manual, `http://www.daimi.au.dk/designCPN`
11. Renew webpage, `renew.de`.
12. Westergaard, M.: Game Coloured Petri Nets. In: Proc. of 7th CPN Workshop. DAIMI-PB, vol. 579, pp. 281–300 (2006)
13. Westergaard, M., Kristensen, L.M.: The Access/CPN Framework: A Tool for Interacting With the CPN Tools Simulator. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 313–322. Springer, Heidelberg (2009)
14. Westergaard, M., Kristensen, L.M., Kuusela, M.: A Prototype for Cosimulating SystemC and Coloured Petri Net Models. In: Proc. of 10th CPN Workshop. DAIMI-PB, vol. 590, pp. 1–19 (2009)
15. 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)

Crocodile: A Symbolic/Symbolic Tool for the Analysis of Symmetric Nets with Bag*

M. Colange[1], S. Baarir[2], F. Kordon[1], and Y. Thierry-Mieg[1]

[1] LIP6, CNRS UMR 7606, Université P. & M. Curie – Paris 6
4, place Jussieu, F-75252 Paris Cedex 05, France
Maximilien.Colange@lip6.fr, Fabrice.Kordon@lip6.fr, Yann.Thierry-Mieg@lip6.fr
[2] LIP6, CNRS UMR 7606 and Université Paris Ouest Nanterre La Défense
200, avenue de la République, F-92001 Nanterre Cedex, France
Souheib.Baarir@lip6.fr

Abstract. The use of high-level nets, such as colored Petri nets, is very convenient for modeling complex systems in order to have a compact, readable and structured specification. Symmetric Nets with Bags (SNB) were introduced to cope with this goal without introducing a burden due to the underlying complexity of the state space. The structure of bags allows through exploitation of symmetries to provide a compact quotient state space representation (similarly to the construction proposed in GreatSPN).

In this paper, we present Crocodile, the first implementation of a modeling environment and model checker dedicated to SNB. Its goal is first to be a proof of concept for experimenting the quotient graph techniques together with hierarchical set decision diagrams. A second objective is to enable experimentation of modeling techniques with this new class of Petri nets.

Keywords: Symmetric Nets with Bags, Model Checking, Symmetries-based techniques, Hierarchical Set Decision Diagrams.

1 Introduction

Symmetric nets with Bags (SNB) [7] are a compact and readable dialect of colored Petri nets allowing structured specification of complex systems. They are based on Symmetric Petri nets (SN), formerly known as Well-Formed Petri Nets [2], a subclass of High-level Petri Nets[1]. Like SN, they allow construction of a quotient state graph representation [8], automatically derived from the specification, that preserves many properties of interest (e.g. LTL), because bags[2] in tokens remain compatible with symmetry-based reduction techniques.

This paper presents the tool Crocodile, which allows creation and analysis of SNB. The definition of SNB is quite recent [7], and this is the first tool that allows their manipulation. It is composed of an Eclipse plugin for front-end modeling (based on the

* Supported by the FEDER Île-de-France/System@tic—free software NEOPPOD project.

[1] "Symmetric Nets" have been chosen in the context of the ISO standardization.

[2] 'Bag' is a synonym for 'multiset'.

L.M. Kristensen and L. Petrucci (Eds.): PETRI NETS 2011, LNCS 6709, pp. 338–347, 2011.

Coloane editor [9]), and it uses hierarchical Set Decision Diagrams (SDD) [5] in the back-end to support construction of the quotient state graph.

The paper is structured as follows. Section 2 informally presents SNB and their relation to SN. Then, section 3 describes the architecture of the tool as well as its original encoding of the quotient graph. Section 4 provides some information about performances of Crocodile.

2 Informal Presentation of Symmetric Nets with Bags (SNB)

This section informally presents SNB. Due to lack of place, we do not introduce the formal definitions that can be found in [7].

The SaleStore example. Let us present SNB by means of a simple example, the SaleStore (see Figure 1). People enter the sale store through an **airlock** with a capacity of two (of course, only a single person may enter too). Then, people may buy items (at most two but possibly zero if none fits their need) and leave with the acquired items. Let us note that this example has two scalable parameters: **P**, the number of involved people in the system and **G**, the number of available gifts in the warehouse.

The model in Figure 1 illustrates most of the main features of SNB. First, there are several color types giving the place's domains: simple color types like *People* or *Gift* are called classes, while bags such as *BagPeople* or *BagGift* and cartesian products such as *PeopleBagGift* are built upon basic color classes.

Variables which are formal parameters for transition binding are declared in the *Var* section. A basic variable such as *p* can be bound to represent any element of *People*. A variable such as *BP* represents a multiset (or bag) of *People*; since it is tagged by the `unique` keyword, it can actually only be bound to a subset of *People* (each element in *BP* appears at most once). Variable *BG* is not tagged with `unique` keyword; it could be bound to any multiset of gifts (if the warehouse was configured to contain several instances of each gift for instance).

Transition guards can be used to constrain the cardinality of a Bag variable : the constraint $[card(BP) < 3$ and $card(BP) > 0]$ on **airlock** model its capacity of at most 2 people (the airlock does not operate empty), while the constraint on **shopping** bounds the number of gifts bought in the store by each person.

Equivalence with Symmetric Nets. SNB have the same expressiveness as Symmetric Nets but allow for more compact modeling. Like colored nets which can be seen as an abbreviation of P/T nets, SNB can also be unfolded into an equivalent SN. Figure 2 shows such an unfolding. Transitions **airlock** and **shopping** are replicated for each

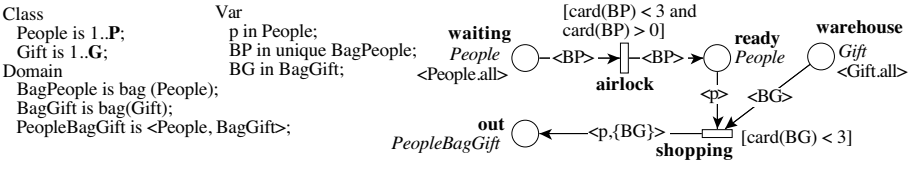


Fig. 1. The SaleStore example modelled with a SNB

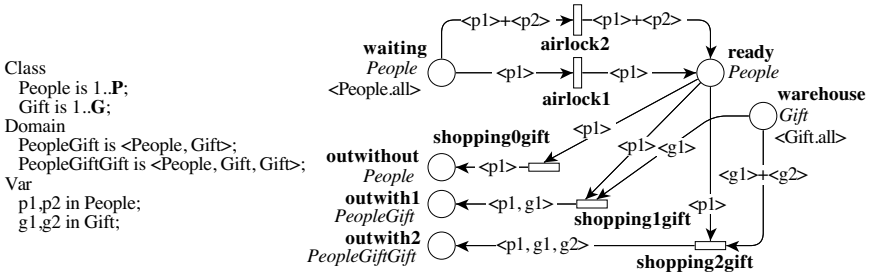


Fig. 2. Unfolding of the SNB presented in Figure 1 into a SN

possible cardinality of the bag variable they can instantiate. Place **out** in the SNB is unfolded according to the various domains obtained by "flattening" the bag token of the *PeopleBagGift* type. Hence, the greater the bound on the cardinality, the more places in the unfolded SN. Note that modeling anything the customers do once they are "out" with their gifts would be very cumbersome in the SN.

However, designers must be careful when defining guards and initial markings so that the model remains finite if unfolded to SN. For example, if **shopping** had no **warehouse** place in input and no guard, an infinite number of bags could be generated (leading to an infinite unfolding). The main advice for the designers is to always bound the cardinality of the bag-variables.

Advantages of SNB. From Figures 1 and 2, it is obvious that with bags manipulation, SNB provide a much more compact and natural way to model system than SN.

Another advantage of SNB is to allow production of a more compact quotient reachability graph in number of edges. This is due to the use of bag variables which better express the symmetries of possible bindings of variables to values. For instance, when choosing two *People* from **waiting**, **airlock** in the SNB allows $P \times (P-1)$ bindings (*i.e.* choose two from P), while **airlock** in the SN allows $2 \times P \times (P-1)$ because variables $p1$ and $p2$ can independently be bound to any element of *People*. Since computing successors in a quotient graph is a costly operation (due to the canonization procedure), this aspect may heavily impact the performance of analysis tools.

Issues in Representing the State Space. Implementation of symmetry-based techniques is not a challenge anymore since tools such as GreatSPN [6] or MurPhi [11] have efficiently implemented such algorithms for over 20 years using explicit data structures.

The challenge resides in combining so-called "symbolic" techniques based on symmetries with the so-called "symbolic" techniques based on decision diagrams. Such a "symbolic/symbolic" approach was first experimented in [13] on top of Data Decision Diagrams (DDD) [4]. Here, the hierarchical structure of SNB for both the net structure, the types and the tokens strongly suggests to take benefits of a new decision diagram structure: Hierarchical Set Decision Diagrams (SDD) [5]. The engine developed to generate the state graph is thus fully based on decision diagrams; note that this differs from [1] which does not support construction of a quotient graph, but tries to deal with performance evaluation of stochastic symmetric nets using decision diagrams.

Our work aims at showing that the two techniques can be combined and provide, in favorable conditions, added gains.

3 Tool Architecture

So far, Crocodile is a "symbolic/symbolic" state space generator for SNB able to compute reachability properties. As already mentioned, its purpose is to provide a first modeling and analysis tool for SNB as well as to merge two well known techniques for efficient state space generation. This section sketches the tool architecture first, and then presents the encoding technique we use for an efficient storage of quotient state graphs.

Use of the Tool. Crocodile is plugged in the Coloane modeler [9] (see Figure 3). It is written in C++ and uses the `libddd` [10] for SDD manipulation.

Coloane is a generic graph editor in which the concrete syntax of SNB has been plugged. Once the model is designed, it is possible to invoke Crocodile directly from the "Coloane Services" menu. Then, a windows requesting for a reachability formula pops out. If no formula is provided, Crocodile simply generates the state space. We use the syntax proposed for the model checking contest [12] that has been extended to support SNB markings. Various statistics can be displayed: number of symbolic states in the quotient state graph, number of SDD nodes, the number of canonizations that have been computed, etc. Assessment and performances (section 4) are computed using a standalone version of the tool under Unix (also distributed).

Symbolic/Symbolic Representation of SNB states. Let us first remind the main characteristics of symbolic markings as they were presented in [3]. The main idea is to avoid representing similar behaviors, *i.e.,* identical behaviors with respect to values permutations. To do so, the actual "identity" of values is forgotten and only their distributions among places are stored. Values with the same distribution and belonging to the same color type are grouped into a so-called *dynamic subclass*. A symbolic marking is, then,

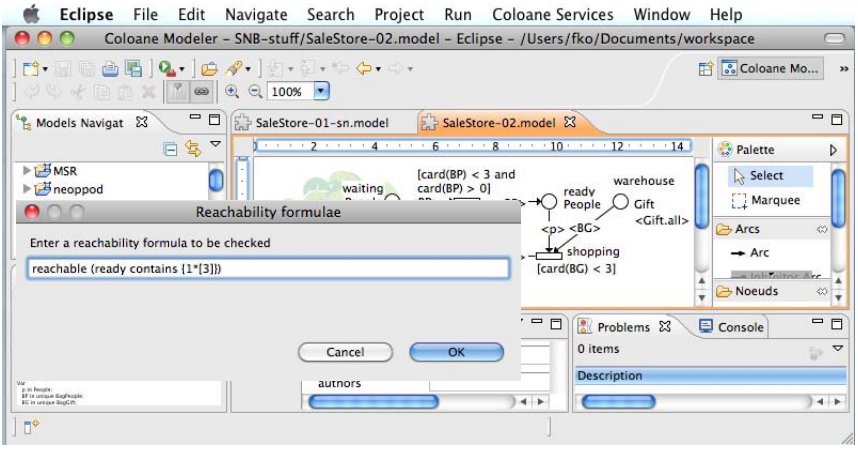


Fig. 3. Crocodile in the Coloane User Interface

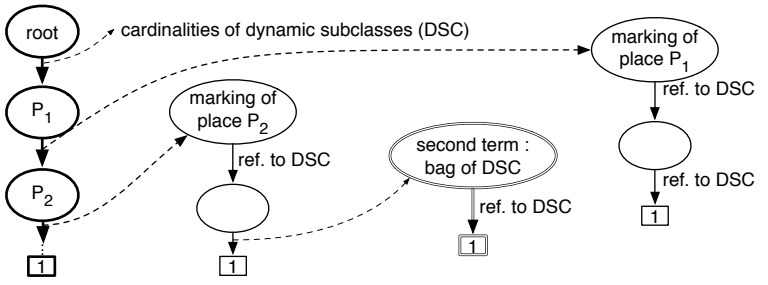


Fig. 4. Architecture of Crocodile encoding of a symbolic marking

a cartesian product of dynamic subclasses and will represent a large number of concrete markings (according to the cardinalities of the involved subclasses).

The originality of Crocodile is to encode these symbolic markings by means of Hierarchical decision diagrams. SDD extend DDD by proposing a way to hierarchically encode data. They also inherit the notion of homomorphism that was defined in DDD [4]. Both computation of successor states and their canonization are implemented as homomorphisms.

Encoding. Roughly, where BDD represent sets of boolean assignments, SDD represent sets of set assignments. The first consequence is that SDD arcs are valued with sets, where BDD arcs are valued with booleans. The second consequence is the hierarchy: since arcs are valued with sets, and SDD themselves represent sets, the arcs of a SDD may refer to another SDD. This increases the sharing capacity of decision diagrams and highlights their hierarchical feature.

Let us illustrate our encoding scheme with Figure 4. A symbolic marking is represented by:

- the identification of dynamic subclasses cardinalities,
- the symbolic content of each place (as a cartesian product of dynamic subclasses).

Our encoding presents three levels. The first one (bold) corresponds to the structure of the SNB and lists its places. The second level (thin) is reached from the arcs between the nodes encoding places and describe their symbolic marking. The third level (double thin) stands to encode bag tokens. As both second and third levels represent bags, they may share a given description (its interpretation is then handled by the homomorphism that operate on the structure). Figure 5 shows an example of this capability for two partial markings (only places **waiting**, **warehouse** and **out** are represented) of the SaleStore example:

$$M_1 = waiting(P_1) + warehouse(G_0) + out(\langle P_0, \{G_1\}\rangle)$$
$$M_2 = waiting(P_1) + warehouse(G_1) + out(\langle P_0, \{G_0\}\rangle)$$

We assume that P_0 and P_1 are dynamic subclasses in *People* while G_0, G_1 are dynamic subclasses in *Gift*.

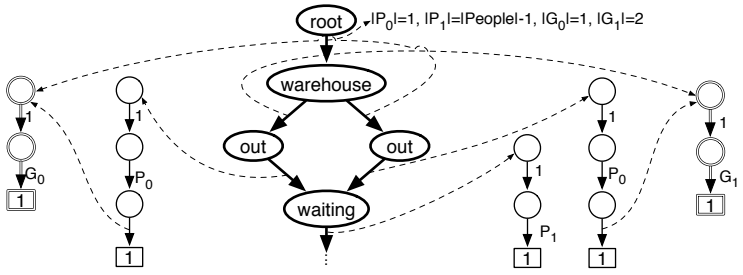


Fig. 5. Example of encoding for two markings with common shared parts

This Figure shows the encoding of these two partial markings. The three levels of hierarchy are clearly visible. We also observe that the markings G_0 (and G_1) are shared at two different levels. The dotted path from the arcs below place **out**, corresponds to a bag inside a bag while, from place **warehouse**, they are simply a bag.

Symbolic/Symbolic Representation of SNB arcs. In most decision diagram-based state space representations, reachability graph arcs are not explicitly stored in memory. Yet, they may be reconstituted when necessary (*i.e.* when elaborating a counter example) through the firing relation. This is also the case in Crocodile. We thus trade memory against CPU when elaborating the counter example.

Summary. Our objective is to stack the two so-called "symbolic" mechanisms.

First, symbolic states allow a compact representation by grouping similar states up to permutations thanks to dynamic subclasses that gather structurally equivalent values in color types.

Second, symbolic encoding of such a state representation allows to share common parts of the description, thus saving memory and providing a fast way to compare symbolic markings. The use of the SNB structure (graph, tokens, bags in tokens) even increases the sharing capacity between levels of representation with SDD.

4 Assessment and Performances

This section shows how we assessed our tool on SNB by using a comparison with GreatSPN [6] that is now the reference implementation of the quotient state graph for SN. In this case, we only use the colored features of GreatSPN that also handles stochastic nets[3]. We also run our tool on both the SN and SNB models of section 2 with various values for the two scaling parameters (size of types *People* and *Gift*).

Assessment. since SNB encompass SN, we use GreatSPN as a comparison when processing SN with Crocodile. In both cases, the size of the state space is the same.

We also observe the same number of states for the quotient state space for both SN and SNB, which is consistent too. In fact, the main difference between the symbolic state space of a SNB and the one of its unfolded SN is the number of symbolic arcs.

[3] Well-Formed Petri Nets [2] introduce stochastic features that are not yet embedded in SNB.

Table 1. Compared performances of Crocodile and GreatSPN on state space generation

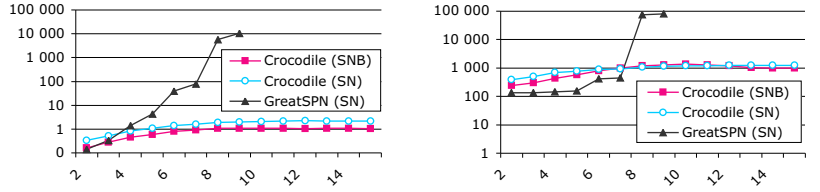
P	G	Number of Nodes		SNB (Crocodile)			SN (Crocodile)			SN (GreatSPN)		
		Quotient Graph	Ordinary Space	# of Firings	Mem. in KB	Time in seconds	# of Firings	Mem. in KB	Time in seconds	# of Firings	Mem. in KB	Time in seconds
5	6	116	6.70×10^{05}	285	502	0.4	361	703	0.7	10 430	407	17.0
5	7	120	2.75×10^{06}	296	602	0.4	377	745	0.7	60 850	433	29.7
5	8	124	1.10×10^{07}	303	684	0.4	388	781	0.8	99 920	71 958	2 041.1
5	9	125	4.18×10^{07}	305	653	0.4	392	784	0.9	3 497 661	77 628	3 128.2
5	10	126	1.51×10^{08}	306	572	0.4	394	783	0.9	—	MOVF	—
5	50	126	4.24×10^{16}	306	573	0.5	394	797	1.1	—	MOVF	—
6	6	180	4.12×10^{06}	496	794	0.7	652	875	1.2	28 612	422	39.8
6	7	190	1.97×10^{07}	527	976	0.8	695	934	1.3	146 775	448	77.7
6	8	200	9.24×10^{07}	550	1 206	0.9	730	1 087	1.6	289 301	73 855	5 857.8
6	9	204	4.22×10^{08}	561	1 277	0.9	745	1 132	1.7	10 589 107	79 526	10 564.5
6	10	208	1.86×10^{09}	568	1 357	1.0	756	1 183	1.8	—	MOVF	—
6	11	209	7.78×10^{09}	570	1 284	0.9	760	1 184	1.8	—	MOVF	—
6	12	210	3.07×10^{10}	571	1 182	0.9	762	1 227	1.9	—	MOVF	—
6	50	210	1.03×10^{19}	571	1 002	1.1	762	1 245	2.1	—	MOVF	—
7	6	260	2.29×10^{07}	744	1 275	1.2	967	1 242	2.0	64 531	453	83.5
7	7	280	1.24×10^{08}	811	1 578	1.5	1 052	1 408	2.4	304 504	880	196.6
7	8	300	6.65×10^{08}	864	1 909	1.8	1 127	1 615	2.9	680 594	75 753	12 024.3
7	9	310	3.19×10^{09}	896	2 135	2.0	1 168	1 734	3.2	22 553 532	81 424	23 548.3
7	10	320	1.81×10^{10}	919	2 151	2.1	1 200	1 985	3.5	—	MOVF	—
7	14	330	8.34×10^{12}	940	1 981	2.0	1 232	2 132	4.0	—	MOVF	—
7	100	330	4.18×10^{27}	940	1 811	2.7	1 232	2 172	4.9	—	MOVF	—
8	6	356	1.19×10^{08}	1 084	1 301	1.8	1448	1 611	3.04	123 639	486	151.86
8	7	390	7.11×10^{08}	1 208	1 967	2.33	1606	1 861	3.73	587 084	948	495.11
8	8	425	4.27×10^{09}	1 312	2 600	3.02	1754	2 240	4.63	1 496 928	77 654	21300.24
8	9	445	2.54×10^{10}	1 382	3 134	3.47	1845	2 379	5.2	40 063 379	83 326	43946.47
8	10	465	1.49×10^{11}	1 436	3 496	3.93	1924	2 531	5.84	—	MOVF	—
8	16	495	2.90×10^{15}	1 511	3 568	4.02	2032	2 653	6.97	—	MOVF	—
8	100	495	3.10×10^{31}	1 511	3 255	5.04	2032	2 742	8.34	—	MOVF	—
20	40	10 626	3.17×10^{51}	41 529	1 401 621	6016.79	57051	1 150 338	5422.91	—	MOVF	—

This will lead to the analysis of less successors for symbolic states and thus, less canonizations (as illustrated in Table 1 and Figure 6).

Performance. To evaluate performance of state space generation (to first verify safety properties), we use the models presented in section 2. The SNB of Figure 1 was processed by Crocodile while its corresponding SN was processed by both Crocodile and GreatSPN. This enables a separate evaluation of the gain brought by the encoding compared to the one coming from the use of bags in tokens.

We let the number of values in *People* and *Gift* increase progressively. Executions were operated on a 32bits 3.2GHz Intel processor with 3GByte of memory and running Linux. Time was measured with `time` and memory estimated using `memusage`. Table 1 summarizes the collected information. Columns show:

- P, the size of class *People*,
- G, the size of class *Gift*,
- the size (number of nodes) for both the quotient state graph and the state space (*i.e.* the corresponding concrete state space),



(a) Computation time in seconds (b) Memory consumption in Kbyte

Fig. 6. Memory and time measures for $|People| = 6$ and $|Gift|$ varying from 2 to 15

– for SNB (with Crocodile) and SN (with both Crocodile and GreatSPN): the number of firings[4] performed to build the quotient state graph, consumed memory[5] and execution time in seconds.

Table 1 clearly shows (once again) the benefits brought by symbolic techniques compared to explicit ones, when models exhibit symmetries.

It also shows the very low number of firings Crocodile needs to build the SNB quotient state graph compared to GreatSPN for the SN one. Crocodile also explore more successors for SN than for SNB, which is related to the reduced number of transitions in the SNB. Both tools must canonize each new discovered state to check if it belongs to an already computed one. Since the canonization algorithm is time consuming, this has a dramatic impact on GreatSPN execution. This impact can be noticed in Figure 6(a) that shows execution time for $|People| = 6$ and $|Gift|$ varying from 2 to 15.

The benefits of decision diagram based representation is also highlighted when comparing GreatSNPN and Crocodile running on SN. For small values of P and G, shared parts of the quotient state graph are not sufficient to overload the initial cost of the decision diagram structure, this becomes false when G ≥ 8 in Table 1, thus leading to consequent gain in memory usage. This is also visible in Figure 6(b) that shows the evolution of memory consumption for $|People| = 6$ and $|Gift|$ varying from 2 to 15.

When G > 9, the combinatorial explosion of firings forces GreatSPN to stop. Thus, we can process the example for large values such as 20 peoples with 40 gifts (last line in the table, asymptote point in the quotient state space for this system configuration).

Let us notice two points for this model. First, the combinatorial explosion dramatically increases every two increments of G. This is due to the maximum bound of gifts to be bought (see guard in transition **shopping**, that bound this value to 2). This increases can be directly observed in the charts of Figure 6. Second, the number of symbolic markings stabilizes when $|Gift| \geq 2 \times |People|$. When $|Gift|$ is just below the stabilization value ($2 \times |People|$), the SDD structure is not fully dense; beyond this value, the sharing in the SDD structure is maximized. Reaching this point of maximal sharing results in a slight decrease of memory consumption for Crocodile.

State Space Snalysis. So far, Crocodile provides analysis of reachability properties. Such properties are constraints that can be checked during state space generation. This

[4] They are symbolic/symbolic firings for Crocodile and symbolic firing for GreatSPN.

[5] MOVF means memory overflow (around 2.03 Gbytes on our experiment machine).

does not bring extra complexity (just a constant due to the property evaluation). Evaluation of a reachability property is done using the following schema:

- translation of the property into constraints c on the symbolic markings (expressed as a SDD),
- for each new symbolic state s, comparing the canonical representation of s with c (since both are SDD, this is a fast operation).

So far, once a state verifying the property is found, the tool must reexecute the state space generation algorithm to store the list of symbolic firings leading to the identified state. Thus, verification of a reachability property may lead to building twice the state space in the worst case. This complexity is compensated by the gain in the state space generation.

Summary. As a conclusion to these experiments, we note the two so-called "symbolic" techniques (the one based on symmetries and the one based on decision diagram encoding) stack well. First, the traditional quotient state graph brings an exponential gain with respect to the ordinary graph. Then, the SDD based encoding brings another exponential gain on top of the previous one. As Figure 6 shows, most of the gains observed on SN are brought by the simultaneous use of these techniques.

We have another confirmation that coupling the two symbolic techniques is of interest. A prototype version of GreatSPN uses several variants of decision diagrams [1]: multi-way DD (MDD), multi-terminal MDD (MTMDD), and edge-valued MDD (EV+ MDD). None of these are hierarchical and they encode Stochastic P/T nets so far. Their results also show significant gain from the original version.

5 Conclusion

This paper presents the tool Crocodile that is original in several manners: *(i)* it is the first implementation of SNB [7] and *(ii)* it encodes the quotient state graph with decision diagrams (symbolic/symbolic approach).

From this work, we can draw three main results. First, SNB show good modeling compacity when manipulating sets or bags in Petri nets. This is illustrated by our small example: the SaleStore model and its unfolding to SN.

Second, as already foreseen in [1], the two so-called "symbolic" techniques (symmetry-based and decision diagram-based) stack very well. Each brings an exponential reduction factor in performances, as shown in section 4. In particular, the hierarchical encoding of markings in the quotient state graph even increases the sharing capacity, thus leading to significant gains in memory.

Third, the theoretical gain in the number of arcs for SNB is experimentally demonstrated. It increases performances already brought by the encoding technique since it simplifies the computation of the quotient state graph (less successors to examine). Moreover, since Crocodile relies on decision-diagrams, we do not explicitly represent arcs, thus increasing memory gain.

Crocodile is available at `http://move.lip6.fr/software/SNB/`. It shows good performances in both memory consumption and execution time. It is able to perform

analysis of reachability properties. However, computation of a counter example might be optimized in the future. So far, it is based on a second computation of the state space.

It would also be of interest to formally define the unfolding from SNB to SN and its reverse operation to be able to enable on-the-fly use of the techniques embedded in Crocodile. We could then cumulate benefits of the SNB model with more classical modeling schemes.

References

1. Babar, J., Beccuti, M., Donatelli, S., Miner, A.: GreatSPN Enhanced with Decision Diagram Data Structures. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 308–317. Springer, Heidelberg (2010)
2. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well-formed coloured nets and their symbolic reachability graph. In: Jensen, K., Rozenberg, G. (eds.) Procedings of the 11th International Conference on Application and Theory of Petri Nets (ICATPN 1990). Reprinted in High-Level Petri Nets, Theory and Application. Springer, Heidelberg (1991)
3. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: A symbolic reachability graph for coloured Petri nets. Theoretical Computer Science 176(1–2), 39–65 (1997)
4. 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)
5. 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)
6. GreatSPN. Petri nets suite, `http://www.di.unito.it/~greatspn`
7. Haddad, S., Kordon, F., Petrucci, L., Pradat-Peyre, J.-F., Trèves, N.: Efficient State-Based Analysis by Introducing Bags in Petri Net Color Domains. In: 28th American Control Conference (ACC 2009), pp. 5018–5025. Omnipress IEEE, St-Louis (2009)
8. Junttila, T.: On the symmetry reduction method for Petri Nets and similar formalisms. PhD thesis, Helsinki University of Technology, Espoo, Finland (2003)
9. MoVe team. The coloane home page, `http://move.lip6.fr/software/COLOANE`
10. MoVe team. The libddd home page, `http://move.lip6.fr/software/DDD`
11. Murphi. Murphi description language and verifier, `http://verify.stanford.edu/dill/murphi.html`
12. SUMO 2011. Sumo model checking contest, `http://sumo.lip6.fr/Model_Checking_Contest.html`
13. Thierry-Mieg, Y., Ilié, J.-M., Poitrenaud, D.: A symbolic symbolic state space representation. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 276–291. Springer, Heidelberg (2004)

Author Index