Lecture Notes in Computer Science 1639
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

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

Susanna Donatelli Jetty Kleijn (Eds.)

Application and Theory of Petri Nets 1999

20th International Conference, ICATPN'99
Williamsburg, Virginia, USA, June 21-25, 1999
Proceedings

Springer

Series Editors

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

Volume Editors

Susanna Donatelli
Università di Torino, Dipartimento di Informatica
C.so Svizzera 185, I-10149 Torino, Italy
E-mail: susi@di.unito.it

Jetty Kleijn
Leiden University, LIACS
2300 RA Leiden, The Netherlands
E-mail: kleijn@wi.leidenuniv.nl

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Application and theory of Petri nets 1999 : 20th international conference ; proceedings / ICATPN '99, Williamsburg, Virginia, USA, June 21 - 25, 1999. Susanna Donatelli ; Jetty Kleijn (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1639)
ISBN 3-540-66132-8

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

ISSN 0302-9743
ISBN 3-540-66132-8 Springer-Verlag Berlin Heidelberg New York

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

© Springer-Verlag Berlin Heidelberg 1999
Printed in Germany

Typesetting: Camera-ready by author
SPIN 10703456 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

This volume contains the proceedings of the 20th International Conference on Application and Theory of Petri Nets. The aim of the Petri net conferences is to create a forum for the dissemination of the latest results in the application and theory of Petri nets. Typically there are some 150-200 participants and usually one third of these come from industry, while the rest are from universities and research institutions. The conferences and a number of other activities are coordinated by a steering committee formed by: G. Balbo (Italy), J. Billington (Australia), C. Girault (France), K. Jensen (Denmark), S. Kumagai (Japan), G. De Michelis (Italy), T. Murata (U.S.A.), C.A. Petri (Germany; honorary member) W. Reisig (Germany), G. Roucairol (France), G. Rozenberg (The Netherlands; chair), M. Silva (Spain).

The 1999 Petri net conference took place in Williamsburg, Virginia, and was organized by the Department of Computer Science of The College of William and Mary, Williamsburg. This was the second time the conference had been organized in the United States. We received 45 submissions from 15 countries on 5 continents of which 21 accepted for presentation. The submitted papers were evaluated by a program committee with the following members: W. van der Aalst (The Netherlands), P. Azema (France), W. Brauer (Germany), S. Christensen (Denmark), A. Desrochers (U.S.A.), S. Donatelli (Italy; co-chair), C. Girault (France), L. Gomes (Portugal), J. Hillston (United Kingdom), E. Kindler (Germany), H.C.M. Kleijn (The Netherlands; co-chair), S. Kumagai (Japan), C. Lakos (Australia), A. Levis (U.S.A.), J. Lilius (Finland), T. Murata (U.S.A.), G. Nutt (U.S.A.), K. Onaga (Japan), W. Penczek (Poland), L. Pomello (Italy), W. Sanders (U.S.A.), M. Silva (Spain), D. Simpson (United Kingdom), P.S. Thiagarajan (India), K. Trivedi (U.S.A.), R. Valette (France), R. Valk (Germany), A. Yakovlev (United Kingdom), W. Zuberek (Canada). The meeting of the program committee took place in Leiden, The Netherlands. In addition to the submitted contributions, there were three invited lectures by P. Varaiya (U.S.A.), B.H. Krogh (U.S.A.), and G. De Michelis (Italy). Other activities before and during the conference included a work-in-progress session, tool presentations and demonstrations, extensive introductory tutorials, two advanced tutorials on Performance Analysis and on Distributed Algorithms, and two workshops on Hardware Design and Petri Nets and on Applications of Petri Nets to Intelligent System Development.

We would like to express our gratitude to all authors of submitted papers and to the members of the program committee and the referees assisting them in providing almost 200 referee reports. We thank M. Boon-van der Nat and D. Costa for their efficient secretarial support. We greatly appreciate the efforts of the organizing committee in Williamsburg, G. Ciardo (chair), E. Smirni, D.S. Noonan, and M.D. Maurer. Also the excellent cooperation with A. Hofmann of Springer-Verlag is gratefully acknowledged.

April 1999 Susanna Donatelli and Jetty Kleijn

List of Referees

J.P. Barros
T. Basten
M. Bednarczyk
L. Bernardinello
G. Berthelot
J. Billington
J. Bradfield
G. A. Bundell
N. Busi
J. Campos
Y. Cao
J.M. Colom
P. De Saqui Sannes
J. Dehnert
K. Drira Ona
C. Dutheillet
J. Esparza
R. Esser
J. Ezpeleta
B. Farwer
C. Ferigato
A. Finkel
V. Galpin
F. Garcia-Valles
M.P. Gervais
H.J.M. Goeman
S. Gordon
P. Gradit
S. Haar
K. Hiraishi
S.J. Huang
N. Husberg
J.M. Ilia
M. Jantzen
K. Jensen
G. Juanole
P. Kemper
A. Kiehn
H. Kinjyo
A. Kondratyev
F. Kordon
M. Koutny
L.M. Kristensen
O. Kummer
A. Martens
O. Matyja
E.W. Mayr
A. Mazurkiewicz
D. Moldt
K.H. Mortensen
M. Mukund
K. Narayan Kumar
M. Nielsen
A. Ohta
J. Padberg
B. Padrin Chezalviel
E. Paviot-Adet
C.J. Perez
S. Peuker
M. Pinna
M. Pitkiewicz Koutny
D. Pointrenaud
J.F. Pradat-Peyre
L. Recalde
W. Reisig
K. Schmidt
S. Shatz
M. Siegle
C. Silbertin-Blanc
C. Simone
M.O. Stehr
Y. Sugasawa
I. Suzuki
S. Takai
E. Teruel
N. Uchihira
Y. Ueda
A. Valmari
K. Varpaaniemi
F. Vernadat
T. Vesper
H. Völzer
M. Voorhoeve
R.F.C. Walters
T. Watanabe
M. Weber
J. Winkoski
H. Yamasaki
Y. Zhou

Table of Contents

Design, Simulation, and Implementation of Hybrid Systems 1
P. Varaiya

Efficient Reachability Set Generation and Storage Using Decision Diagrams 6
A. S. Miner and G. Ciardo

Structural Methods to Improve the Symbolic Analysis of Petri Nets 26
E. Pastor, J. Cortadella, and M.A. Peña

Stubborn Sets for Standard Properties 46
K. Schmidt

Petri Net Based Behavioural Specification of CORBA Systems 66
R. Bastide, P. Palanque, O. Sy, D.-H. Le, and D. Navarre

Symmetric Communication between Coloured Petri Net Simulations and Java-Processes 86
O. Kummer, D. Moldt, and F. Wienberg

Recent Developments in Modeling and Analysis of Hybrid Dynamic Systems106
B. H. Krogh

Autonomous Continuous P/T Systems 107
L. Recalde, E. Teruel, and M. Silva

An Approach to the Analysis of Interworking Traders 127
A. Tokmakoff and J. Billington

Parallel Approaches to the Numerical Transient Analysis of Stochastic Reward Nets 147
S. Allmaier and D. Kreische

SWN Nets as a Framework for the Specification and the Analysis of FT Techniques Adopted in Electric Plant Automation 168
L. Capra, R. Gaeta, and O. Botti

Monitoring Discrete Event Systems Using Petri Net Embeddings 188
C. N. Hadjicostis and G.C. Verghese

Quasi-Static Scheduling of Embedded Software Using Equal Conflict Nets . 208
M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli

Theoretical Aspects of Recursive Petri Nets 228
S. Haddad and D. Poitrenaud

Petri Net Theory - Problems Solved by Commutative Algebra 248
C. Schneider and J. Wehler

Testing Undecidability of the Reachability in Petri Nets with the Help of 10th Hilbert Problem 268
P. Chrząstowski-Wachtel

Net Theory and Workflow Models 282
G. De Michelis

Concurrent Implementation of Asynchronous Transition Systems 284
W. Vogler

Trace Channel Nets 304
J. Fanchon

Reasoning about Algebraic Generalisation of Petri Nets 324
G. Juhás

The Box Algebra - A Model of Nets and Process Expressions 344
E. Best, R. Devillers, and M. Koutny

Detection of Illegal Behaviours Based on Unfoldings 364
J.-M. Couvreur and D. Poitrenaud

Five Classes of Invariant-Preserving Transformations on Colored Petri Nets 384
T.-Y. Cheung and Y. Lu

Verifying Intuition - ILF Checks DAWN Proofs 404
T. Baar, E. Kindler, and H. Völzer

Author Index 425

Design, Simulation, and Implementation of Hybrid Systems*

Pravin Varaiya

Department of Electrical Engineering & Computer Science, University of California
Berkeley, CA 94720, U.S.A
varaiya@eecs.berkeley.edu

Abstract. This talk will present two detailed examples of hybrid systems, covering design, simulation and implementation. The first concerns an Automated Highway System. The second application deals with a collection of autonomous unmanned aircraft. The paper provides a background about hybrid systems.

1 What Is a Hybrid System

A *hybrid system* or *automaton* comprises two interacting components: one component evolves in continuous time and has state variables $x \in R^n$, the other is event-driven and has finitely many states $q \in Q$. With each discrete state q is associated a differential inclusion $\dot{x} \in F(q, x)$. Here $F(q, x) \subset R^n$. Associated with each pair (q, q') is an "enabling zone" or "guard" $G(q, q') \subset R^n$ (if the guard is empty the discrete transition $q \to q'$ is infeasible), and a "reset" relation $R(q, q') \subset R^n \times R^n$.

Suppose the system starts in state (q_0, x_0) at time t_0. The continuous state evolves during the time interval $[t_0, t_1)$ according to the inclusion

$$\dot{x}(t) \in F(q_0, x(t)),\ x(t_0) = x_0$$

until a time t_1 is reached when $x(t_1-) \in G(q_0, q_1)$ for some q_1. Because the guard of the transition $q_0 \to q_1$ is now satisfied, the discrete state changes instantaneously from q_0 to q_1, and the continuous state is reset from $x(t_1-)$ to x_1 where $(x(t_1-), x_1) \in R(q_0, q_1)$. (If another guard is satisfied following the reset, there may be another discrete transition.) The continuous state now evolves during $[t_1, t_2)$ according to

$$\dot{x}(t) \in F(q_1, x(t)),\ x(t_1) = x_1$$

until at t_2 another guard is satisfied and the system makes a discrete transition and resets the continuous state as before.

Thus a system trajectory evolves in two phases. In the first phase, the discrete state is unchanged, time progresses and the continuous state evolves. In the

* Research supported by National Science Foundation and Office of Naval Research.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 1–5, 1999.
© Springer-Verlag Berlin Heidelberg 1999

second phase, time stops, the discrete state makes a transition and the continuous state is reset. In this definition, the hybrid system is non-deterministic: the differential inclusion may permit more than one continuous state trajectory; and more than one guard may be satisfied, permitting a choice of discrete transition. Additional conditions on F, G and R are needed to ensure the existence of a trajectory, its continuation for arbitrarily large time, or to make the system deterministic.

Hybrid system models are appropriate in several contexts. In one context a computer or human symbol-processing system provides high level supervision of a continuous process.[1] An example is an automated vehicle whose maneuvers are planned and coordinated at a logical level but executed using continuous time feedback laws that determine the throttle, braking, and steering actuator signals [1]. Another example is in our daily use of "common sense" rules to control sophisticated processes such as microwave ovens, automobiles and copying machines, without understanding the physical and engineering principles that govern the behavior of those processes. A different context is that of "switched" dynamical systems characterized by differential equations whose right-hand side changes discontinuously when the state reaches a switching surface. An automatic gear shift vehicle would be described by such a switched differential equation.[2]

2 Decidability and Verification

The *reach set* $R(q, x, t)$ of a hybrid system is the set of all states in $Q \times R^n$ that can be reached at time t, starting in state (q, x) at time 0. Many properties of system behavior can be formulated in terms of the reach set. For instance, "safety" means that a specified "unsafe" set U is never reached, i.e. $R(q_0, x_0, t) \cap U = \emptyset$ for every t. And "liveness" means that a set S can be reached infinitely often, i.e. $R(q_0, x_0, t) \cap S \neq \emptyset$ for arbitrarily large t. Thus the ability to calculate the reach set permits one to evaluate many properties. The hybrid system is said to be *decidable* if there is a finite algorithm for computing its reach set, it is *undecidable* if there is no such algorithm.

Most hybrid systems are undecidable, which is not unexpected. The fundamental result here is that *timed* automata are decidable [2]. These are hybrid automata with the simplest inclusions: $\dot{x}_i = 1$, (so that the n continuous variables x_i all are 'timers' or 'clocks'); the guards all are conjunctions or disjunctions of the form $x_i \succ c$ where $\succ$ is $>, \geq, <, \leq$ and c is an integer; and if a continuous state variable is reset, it is reset to the value 0. Timed automata are used in applications to express real-time restrictions. The fundamental result implies that

[1] Indeed in computer science, it is customary to refer to the discrete state as a (control) location, indicating its supervisory status.

[2] About 80 percent of the control code for highly automated systems such as a Flight Control System is now concerned with logic (discrete transitions, guards, resets) and 20 percent deals with continuous states (differential inclusion). Hybrid system models are essential if one wishes to fully describe these control systems.

properties of a digital circuit or real-time control algorithm expressed as a timed automaton can be automatically verified.[3]

The decidability of timed automata would be unsurprising if time were discrete, rather than continuous. It was shown in [3] that if all guards are closed (i.e. only $\geq$ and $\leq$ are permitted), then time can be discretized. In [4] it is shown that discretization is possible even when guards are open, although the construction is no longer intuitive.

A *multirate* automaton is a generalization of a timed automaton in which the timers may have different rates: $\dot{x}_i = r(q)$ in discrete state q, where $r(q)$ is an integer not necessarily equal to unity. That there is something special about timed automata is suggested by examples of multirate automata that are undecidable. Much work has been devoted to delineating multirate automata that *are* decidable.

An important class of decidable hybrid systems are the *rectangular* automata. These are systems in which the inclusions $F(q, x)$, the guards $G(q, q')$ and the resets $R(q, q')$ are all rectangles, i.e. of the form $\prod[a_i, b_i]$ with integer coordinates. In particular, the differential inclusions impose only an upper and lower bound on the velocity of each coordinate (the bounds may depend on the discrete state)

$$\dot{x}_i \in [l_i(q), u_i(q)], \ i = 1, ..., n.$$

A rectangular system is *initialized* if in each discrete transition $(q \rightarrow q')$ that leads to a change in the inclusion of a component of the continuous state (i.e. $\{l_i(q), u_i(q)\} \neq \{l_i(q'), u_i(q')\}$), that component is reset during the transition. The main result of [5] is that initialized rectangular automata are decidable. The whole issue of decidability of hybrid automata is thoroughly explored in [6].

The next question is: How general are (initialized) rectangular automata? One answer developed in [8] is that rectangular automata can be used to upperbound the behavior of general hybrid systems, which permits verification of safety properties. The second answer is that it is possible to approximate any nonlinear inclusion arbitrarily closely over any *finite* interval by a rectangular automaton [9]. It is important to extend this approximation result over the infinite time interval.

Software tools, based on these decidability results, have been developed to verify control system designs [10,11].

3 Simulation

SHIFT is a programming language for simulating a network of interacting hybrid systems [12]. In SHIFT hybrid systems can interact in one of three ways. First,

[3] The annual workshop on Computer Aided Verification provides a forum for results in this area and the annual Hybrid Systems workshop serves an analogous function. Papers presented at these workshops are published in the LNCS series of Springer. In addition, the annual Conference on Decision and Control now features one stream of sessions devoted to hybrid and discrete event systems. The February 1998 issue of the Transactions on Automatic Control is dedicated to hybrid systems.

outputs of one system can be inputs of another. Second, the discrete transitions in several systems can be synchronized. Third, the interactions among systems can change dynamically: for example, the input-output connections can be made to depend on the state of the overall system. These facilities for describing hybrid systems and their interactions make SHIFT a very flexible and powerful language for describing hybrid systems. SHIFT is object oriented which makes it well-suited for developing applications in teams. Several other features (type checking, debugging) make it easy to learn.

SHIFT has been extensively used in the detailed simulation of hundreds of interacting automated vehicles. The SHIFT simulation environment is available free and runs on UNIX and NT platforms [13]. Tools for verification of some SHIFT programs are being developed.

4 Implementation

The synchronization procedures of SHIFT make it a very powerful language for simulating complex systems. However, these procedures cannot be realized in real time. A recent commercial system, Teja, not only provides SHIFT-like high-level constructs for designing hybrid system controllers, but it also automatically generates code for real-time control. Teja provides a visual environment for modeling the application, a platform for simulation and testing the real-time performance, a facility for managing a distributed application, and a real-time execution platform [14].

References

1. P. Varaiya, "Smart cars on smart roads: Problems of control," *IEEE Transactions on Automatic Control*, vol. 38, pp. 195–207, February 1993.
2. R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *Proceedings 17th ICALP: Automata, languages and programming*, vol. 443, Lecture Notes in Computer Sciences, Springer, 1990.
3. T. Henzinger, Z. Manna, and A. Pnueli, "What good are digital clocks?," in *ICALP 92: Automata, Languages, and Programming* (W. Kuich, ed.), Lecture Notes in Computer Science 623, pp. 545–558, Springer-Verlag, 1992.
4. A. Gollu, A. Puri, and P. Varaiya, "Discretization of timed automata," in *Proceedings of the 33rd Conference on Decision and Control*, (Lake Buena Vista, FL), December 14-16 1994.
5. A. Puri and P. Varaiya, "Decidability of hybrid systems with rectangular inclusions," in *Computer Aided Verification, CAV '94* (D. Dill, ed.), vol. LNCS 818, pp. 95–104, Springer, 1994.
6. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?," in *Proc. 27th Symposium on the Theory of Computing*, pp. 373–382, 1995.
7. A. Puri and P. Varaiya, "Driving safely in smart cars," in *Proceedings of the American Control Conference*, pp. 3597–3599, June 1995.
8. A. Puri and P. Varaiya, "Verification of hybrid systems using abstraction," in *Hybrid Systems II*, pp. 359–369, LNCS 999, Springer, 1995.

9. A. Puri and P. Varaiya, "ϵ-approximation of differential inclusions," *Hybrid Systems III: Verification and Control*, vol. LNCS 1066, pp. 362–376, 1996.
10. C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The tool KRONOS," *Hybrid Systems III: Verification and Control*, vol. LNCS 1066, pp. 208–219, 1996.
11. T. Henzinger and P.-H. Ho, "HyTech: The Cornell Hybrid Technology Tool," in *Hybrid Systems II*, pp. 265–294, LNCS 999, Springer, 1995.
12. A. Deshpande, A. Gollu, and L. Semenzato, "The SHIFT programming language and run-time system for dynamic networks of hybrid systems." *IEEE Transactions on Automatic Control*, vol. 43, pp. 584–587, February 1998.
13. See: `http://www-shift.eecs.berkeley.edu/downloads.html` for the NT version and `http://www.path.berkeley.edu/shift` for the UNIX version, tutorials and examples.
14. See `http://www.teja.com`

Efficient Reachability Set Generation and Storage Using Decision Diagrams

Andrew S. Miner* and Gianfranco Ciardo

Department of Computer Science
College of William and Mary
Williamsburg, VA, 23185
{asminer,ciardo}@cs.wm.edu

Abstract. We present a new technique for the generation and storage of the reachability set of a Petri net. Our approach is inspired by previous work on Binary and Multi-valued Decision Diagrams but exploits a concept of locality for the effect of a transition's firing to vastly improve algorithmic performance. The result is a data structure and a set of manipulation routines that can be used to generate and store enormous sets extremely efficiently in terms of both memory and execution time.
Classification: Reachability set generation. System verification. Computer tools.

1 Introduction

The generation of the state space, or reachability set, $\mathcal{S}$ for a discrete-state model is an essential step in many types of studies. In the case of logical verification, the goal might be to ensure that no "bad" states satisfying certain boolean conditions can be reached from the initial state. In the case of Markov stochastic modeling, the reachable states determine the size and meaning of the probability vector computed as a result of a numerical solution. In either case, Petri nets (or stochastic Petri nets) are often the formalism of choice to describe such discrete-state models in a formal and compact way.

For the type of problems we consider, $\mathcal{S}$ is finite but its size is so large that its exploration and storage become formidable challenges. Implementors of computer tools for the analysis of stochastic Petri nets quickly found out that the effort to generate $\mathcal{S}$ and the infinitesimal generator matrix $\mathbf{Q}$ of the underlying model was often comparable to that of the numerical solution of the Markov chain. Analogously, $\mathcal{S}$ could easily require an amount of storage comparable to that of $\mathbf{Q}$. In practice, these tools could manage reachability sets ranging in size from 10^4 to perhaps 10^6 states, depending on the quality of the implementation and on the amount of primary memory available.

To move beyond these limitations, innovative approaches had to be employed. In Sect. 2, we briefly present two lines of work that are related to, and indeed inspired, our research. One is represented by the 1994 paper by Pastor et al. [17],

* A.S. Miner's work was supported by fellowships from the NASA Graduate Student Research Program and the Virginia Space Grant Consortium

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 6–25, 1999.
© Springer-Verlag Berlin Heidelberg 1999

who proposed the use of Binary Decision Diagrams (BDDs) [2,3] for the storage and generation of the reachability set for a safe (1-bounded) Petri net. By exploiting the compact representation of BDDs, they generated very large reachability sets in a matter of hours, with small storage requirements. In later work [15,16], a more efficient encoding is introduced based on place invariants; however, the underlying logic is still based on binary variables. In our work, we instead use an extension of BDDs to non-binary logic, as proposed for the Multi-valued Decision Diagrams (MDDs) of Kam [11,19], or the "shared trees" of Zampunièris [20]. Our title uses the term "decision diagrams" because the results we present, while particularly relevant to MDDs, apply also to BDDs.

We are also building upon our own previous work on state space storage [8]. By using a multilevel data structure based on a decomposition of a Petri net into submodels, we showed how $\mathcal{S}$ can be stored using little over a small integer (i.e., one or two bytes) per state. While this amount is linear in $|\mathcal{S}|$ (unlike the results achieved by Pastor with BDDs, which are sublinear for most models of interest), the reason for introducing our structure was that it fulfills many of the needs of numerical approaches based on Kronecker algebra [10,12]: for this application, we have shown how it substantially improves the solution time for very large and sparse problems [5]. Furthermore, for exact numerical solution, one or two bytes per state represents a small fraction of the overall memory requirements, which include several single- or double-precision vectors of size $|\mathcal{S}|$. Another idea exploited in [8] was that of *event locality*: the realization that we can automatically detect *a priori* the identity of the submodels affected by a given event, so that only the corresponding "local substates" change when the event occurs. This concept of locality, appropriately extended when applied to MDDs, enables us to achieve great speedups.

Another related work is [4], which also considers a state space $\mathcal{S}$ defined as a subset of the cross-product $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ of arbitrary sets. Starting from the representation of $\mathcal{S}$ as a boolean vector of size $|\mathcal{S}_K| \cdots |\mathcal{S}_1|$ suggested in [13], common bit subvectors are merged. However, the intent in [4] is the Kronecker-based numerical solution of stochastic Petri nets, so no attempt is made to generate state spaces of the size considered in [17].

In the theoretical results of Sect. 3, we combine the main idea of BDDs (merging common subtrees in the data structure used to encode $\mathcal{S}$) with our idea of locality (limiting the computation and effect of a given event to the affected submarkings) and of model decomposition (defining a marking as a collection of submarkings, not necessarily from a safe net, encoded as a vector of small integer indices), to obtain a very efficient approach for the storage and generation of $\mathcal{S}$.

Sect. 4 explores some implementation issues that affect memory and time complexity. In Sect. 5, we apply our approach to various models previously considered in the literature. For the two applications reported in [17], our approach is much faster and more memory-efficient, hence it can generate much larger states spaces (9.18×10^{626} and 7.29×10^{62} states, respectively).

Finally, Sect. 6 states our conclusions and future research directions.

2 Background and Related Work

Our definition of Petri net is quite general, admitting inhibitor arcs, transition guards, and marking-dependent arc multiplicities [6]. Since any of these extensions achieves Turing-equivalence, we assume that the Petri net model has a finite reachability set $\mathcal{S}$. Formally, we represent a Petri net as a tuple $\left(\mathcal{P}, \mathcal{T}, I, O, H, g, \mathbf{m}^{[0]}\right)$ where:

- $\mathcal{P} = \{p_1, ..., p_{|\mathcal{P}|}\}$ is a finite set of places. A marking $\mathbf{m} \in \mathbb{N}^{|\mathcal{P}|}$ assigns a number of tokens to each place.
- $\mathcal{T} = \{t_1, ..., t_{|\mathcal{T}|}\}$ is a finite set of transitions, with $\mathcal{P} \cap \mathcal{T} = \emptyset$.
- $I : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, $O : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$, and $H : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ describe the marking-dependent multiplicities of the input, output, and inhibitor arcs.
- $g : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \{0, 1\}$ describes the transition guards.
- $\mathbf{m}^{[0]} \in \mathbb{N}^{|\mathcal{P}|}$ is the initial marking.

A transition t is enabled in $\mathbf{m}$, we write $\mathit{Enabled}(t, \mathbf{m})$, iff

$$g(t, \mathbf{m}) = 1 \ \wedge \quad \forall p_i \in \mathcal{P}, \quad I(p_i, t, \mathbf{m}) \leq \mathbf{m}_i \ \wedge \ H(p_i, t, \mathbf{m}) > \mathbf{m}_i.$$

Firing an enabled transition t in $\mathbf{m}$ leads to $\mathbf{n} = \mathit{New}(t, \mathbf{m})$, satisfying

$$\forall p_i \in P, \quad \mathbf{n}_i = \mathbf{m}_i - I(p_i, t, \mathbf{m}) + O(p_i, t, \mathbf{m}).$$

The reachability set $\mathcal{S}$ is then defined as the smallest subset of $\mathbb{N}^{|\mathcal{P}|}$ that contains $\mathbf{m}^{[0]}$ and is closed under the one-step reachability relation; that is, if $\mathbf{m} \in \mathcal{S}$, $\mathit{Enabled}(t, \mathbf{m})$, and $\mathbf{n} = \mathit{New}(t, \mathbf{m})$, then $\mathbf{n} \in \mathcal{S}$ as well.

Since our goal is the efficient generation and storage of $\mathcal{S}$, we briefly describe the traditional approach. An algorithm that iteratively builds $\mathcal{S}$, by processing one marking at a time, is shown in Fig. 1. Its execution time is then at least $O(|\mathcal{S}| \cdot |\mathcal{P}| \cdot |\mathcal{T}|)$. Indeed, it can be worse, since statement 8 requires to search for a newly generated state $\mathbf{n}$ in the set of currently known states. If $\mathcal{S}$ and $\mathcal{U}$ are stored using some type of search tree (e.g., AVL or Splay trees [7] or B-trees [1]), this implies an additional $\log |\mathcal{S}|$ factor.

Considering now the memory usage, a simple approach is to store $\mathcal{S}$ as a search tree where the keys are markings encoded as vectors of $|\mathcal{P}|$ natural numbers (or booleans, for safe nets). However, this requires $O(|\mathcal{P}| \cdot |\mathcal{S}|)$ bytes. Sparse techniques can be used to store the marking vectors, but these are only moderately beneficial, and only when most markings have many zero entries.

2.1 A Multilevel Data Structure to Store $\mathcal{S}$

Given a Petri net, we can partition its set of places $\mathcal{P}$ into K subsets $\mathcal{P}_k$, $k = K, \ldots, 1$, and define $\mathcal{S}_k$ to be the set of reachable "local" submarkings for $\mathcal{P}_k$:

$$\mathcal{S}_k = \{\mathbf{m}_k : \exists \mathbf{m}_K, \ldots, \mathbf{m}_{k+1}, \mathbf{m}_{k-1}, \ldots, \mathbf{m}_1, [\mathbf{m}_K, \ldots, \mathbf{m}_1] \in \mathcal{S}\}.$$

```
Explore(P, T, I, O, H, g, m^[0]);
 1. S ← ∅;                                  • S: markings explored so far
 2. U ← {m^[0]};                            • U: markings found but not yet explored
 3. while U ≠ ∅ do
 4.     choose a marking m in U;
 5.     move m from U to S;
 6.     for each transition t such that Enabled(t, m) do
 7.         n ← New(t, m);
 8.         if n ∉ S ∪ U then
 9.             U ← U ∪ {n};
10. return S;
```

Fig. 1. A procedure *Explore* to generate $\mathcal{S}$.

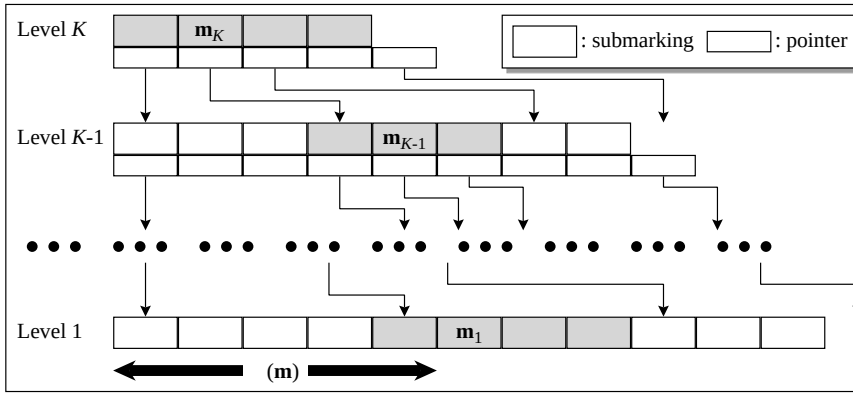


Fig. 2. An array implementation of the multilevel data structure in [8].

In [8], we discussed a data structure to store a subset $\mathcal{S}$ (the *actual state space*) of a cross-product $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ of K sets (the *potential state space*). Fig. 2 illustrates the idea using arrays. At the top level, there is (at most) one instance of each submarking $\mathbf{m}_K \in \mathcal{S}_K$, and the corresponding pointer points to the submarkings for the remaining levels that can coexist with $\mathbf{m}_K$, that is, all $[\mathbf{m}_{K-1}, \ldots, \mathbf{m}_1]$ such that $[\mathbf{m}_K, \ldots, \mathbf{m}_1] \in \mathcal{S}$, and so on. To determine whether a given marking $\mathbf{m} = [\mathbf{m}_K, \ldots, \mathbf{m}_1]$ is reachable, we search for $\mathbf{m}_K$ in the grayed-out portion of level K. If found, we follow the pointer identifying the grayed-out portion of level $K-1$, search that portion for $\mathbf{m}_{K-1}$, and repeat until either we find $\mathbf{m}_1$ in the grayed-out portion of level 1, or we fail to find a submarking $\mathbf{m}_k$, for some $k = K, \ldots, 1$. In the former case, the marking $\mathbf{m}$ is reachable and the offset of the position where we found $\mathbf{m}_1$ in the array for level 1 indicates the lexicographic order $\Psi(\mathbf{m})$ of $\mathbf{m}$ in $\mathcal{S}$ (i.e., the number of reachable markings smaller than $\mathbf{m}$). In the latter case, $\mathbf{m}$ is not reachable.

In practice, it is much more efficient to use the index of a submarking in the data structure encoding $\mathcal{S}$, not the actual submarking, hence only $\lceil \log_2 |\mathcal{S}_k| \rceil$ bits are required for each submarking stored at level k. Assuming a sufficiently large branching factor from each level to the next, the memory requirements are

dominated by the bottom level, for which no pointers are required. Thus, we can store $\mathcal{S}$ in little over $|\mathcal{S}| \cdot \lceil \log_2 |\mathcal{S}_1| \rceil$ bits.

Arrays are used in [8] to reduce memory requirements *after* having completed the exploration of $\mathcal{S}$. However, *during* the exploration, the same approach requires a dynamic data structure allowing efficient search and insertion (e.g., an AVL or Splay tree) for each grayed-out portion at each level; we stress that, in this case, *a priori* knowledge of the sets $\mathcal{S}_k$ is not required, although a decision must be made about an upper bound on their size, so that indices can be stored. In practice, one, two, or four bytes are used (four bytes being rarely needed, as most local reachability sets contain no more than 2^8 or 2^{16} elements).

Another concept from [8] that inspired our current work is the *locality* of a transition. By examining the Petri net definition, it is possible to determine (conservatively) what places are affected when transition t fires. Then, we define the locality k^* of t as the largest k such that $\mathcal{P}_k$ contains an affected place, and we know that any submarking $\mathbf{m}_k$, for $k = K, \ldots, k^* + 1$ is not affected by t.

By exploiting locality we achieve greater efficiency during both exploration of $\mathcal{S}$ and subsequent searches (needed, for example, in a Kronecker-based Markov analysis). If $\mathbf{n} = \mathit{New}(t, \mathbf{m})$, and we know the K pointers, or offsets into the K arrays, $\Psi_K(\mathbf{m}_K), \Psi_{K-1}(\mathbf{m}_K, \mathbf{m}_{K-1}), \ldots, \Psi_1(\mathbf{m}_K, \ldots, \mathbf{m}_1) = \Psi(\mathbf{m})$ for $\mathbf{m}$, the search for $\mathbf{n}$ into our multilevel data structure can start at level k^*, since we know that $\mathbf{n}_k = \mathbf{m}_k$, hence $\Psi_k(\mathbf{n}_K, \ldots, \mathbf{n}_k) = \Psi_k(\mathbf{m}_K, \ldots, \mathbf{m}_k)$, for $k = K, \ldots, k^* + 1$.

2.2 Using BDDs to Store $\mathcal{S}$

It is well-known that a boolean function $f : \{0,1\}^n \to \{0,1\}$ can be represented as a boolean vector of size 2^n indexed starting at zero, where the entry in position $b_n \cdots b_1$, interpreted as an unsigned n-bit integer in base 2, is 0 iff $f(b_n, \ldots, b_1) = 0$. However, this representation requires exponential space. BDDs were introduced [14] to alleviate this problem, as they allow one to encode and compute many boolean functions of interest in a very compact way.

A binary decision diagram (BDD) is a directed, acyclic graph with terminal nodes, labeled from the set {0,1}, and non-terminal nodes, labeled from the set of variable names. Only non-terminal nodes have outgoing arcs, and each non-terminal node has exactly two outgoing arcs, labeled 0 and 1. Every non-terminal node in the graph represents some logic function f. A non-terminal node is represented by the tuple $(x, f_{x=0}, f_{x=1})$, where x is the variable name, the 0 and 1 arcs point to the cofactors $f_{x=0}$ and $f_{x=1}$, respectively (given a function f on variables $x_1, \ldots, x_n$, the cofactors of f with respect to x_i are $f_{x_i=c} = f(x_n, \ldots, x_{i+1}, c, x_{i-1}, \ldots, x_1)$).

An ordered BDD (OBDD) has a total ordering on the variables such that any path of the graph must visit variables in that order. Finally, a reduced OBDD (ROBDD) has the following properties: there are at most two terminal nodes, with labels 0 and 1, there is no non-terminal node $(x, f_{x=0}, f_{x=1})$ with $f_{x=0}$ equal to $f_{x=1}$, and all non-terminals are unique, i.e., there are no two nodes $(x, f_{x=0}, f_{x=1})$ and $(x, g_{x=0}, g_{y=1})$ where $(f_{x=0} = g_{x=0}) \wedge (f_{x=1} = g_{x=1})$. Given a total ordering on the variables, ROBDDs are a canonical representation: two

```
BDDexplore(P, T, I, O, H, g, m^[0]);
1. O ← ∅;                                  • O: old reachability set
2. S ← {m^[0]};                            • S: current reachability set
3. while S ≠ O do
4.     O ← S;
5.     S ← Δ(S) ∪ S;
6. return S;
```

Fig. 3. A BDD-based procedure to generate $\mathcal{S}$.

logic functions f and g are equivalent iff f and g are represented by the same ROBDD. Bryant [2,3] showed how ROBDDs can be efficiently manipulated. In the following we use "BDD" to mean "ROBDD".

Pastor et al. [15,16,17] use BDDs for the generation and storage of the reachability set of a safe Petri net. In [17], the authors partition $\mathcal{P}$ into $K = |\mathcal{P}|$ subsets, with each subset containing a single place. In this case, each place can contain at most one token, so the simple encoding of a single BDD variable per place is used. A BDD is used to encode $\chi_{\mathcal{S}}$, the characteristic function of $\mathcal{S}$, defined by $\chi_{\mathcal{S}}(\mathbf{m}_{|\mathcal{P}|}, \ldots, \mathbf{m}_1) = 1$ iff $\mathbf{m} \in \mathcal{S}$. Hence, we can talk equivalently of boolean functions or sets. The main result of [17] is that, given a set of markings $\mathcal{X}$, we can compute the BDD for the set $\Delta(\mathcal{X})$ of markings reachable from them in one firing, where the operator Δ is itself encoded as a BDD and can be built based on the Petri net definition. Fig. 3 illustrates the idea (we show a simplified version of the algorithm in [17]). Of course, the sets $\mathcal{O}$, $\mathcal{S}$, and $\Delta(\mathcal{S})$ are all encoded as BDDs. More sophisticated partitions and encodings are discussed in [15,16], where invariants are used to reduce the number of boolean variables.

A fundamental property ensuring an efficient approach is that the number of iterations performed by *BDDexplore* is bounded by the *sequential depth* of the Petri net, that is, the maximum number of firings required to reach any marking starting from $\mathbf{m}^{[0]}$ (a quantity no larger than the diameter of the reachability graph). Thus, while each iteration usually implies a substantial computation, the number of iterations is usually quite small.

When the Petri net is not safe, the authors suggest two obvious ways to encode the number of tokens $\mathbf{m}_i$ of place p_i using booleans. If place p_i is k-bounded, we can use a *one-hot encoding* with k variables $b_i^1, \ldots, b_i^k$, at most one of which will be nonzero in any marking (they are all zero iff p_i is empty), or a *binary encoding* with $\lceil \log_2(k+1) \rceil$ variables. The former results in more boolean variables overall, but also in a simpler encoding of the Δ function.

2.3 Multi-valued Decision Diagrams

BDDs have been generalized to integer functions on integer (instead of binary) variables, resulting in MDDs [11,19] (see also the "shared trees" in [20]). MDDs can then represent functions of the form

$$\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n \to \{0, \ldots, m-1\},$$

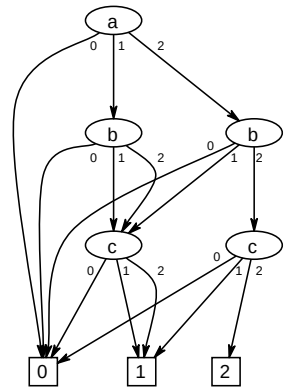


Fig. 4. MDD representing $min(a, b, c)$

```
Case(F, G^0, ..., G^{m-1})
 1. if F is a constant r then return G^r;
 2. if G^0 = G^1 = ··· = G^{m-1} then return G^0;
 3. if G^0 = 0 ∧ ··· ∧ G^{m-1} = m − 1 then return F;
 4. if cache contains entry for (F, G^0, ..., G^{m-1}) then
 5.     return cache entry result;
 6. let x_k be the top variable of F, G^0, ..., G^{m-1};
 7. for i ← 0 to N_k − 1 do
 8.     H^i ← Case(F_{x_k=i}, G^0_{x_k=i}, ..., G^{m-1}_{x_k=i});
 9. if H^0 = H^1 = ··· = H^{N_k−1} then
10.     R ← H^0;
11. else
12.     R ← UniqueTableInsert(x_k, H^0, ..., H^{N_k−1});
13. add [(F, G^0, ..., G^{m-1}), R] to cache;
14. return R;
```

Fig. 5. Algorithm for the *Case* operator

where $\mathcal{S}_i = \{0, \ldots, N_i - 1\}$. Non-terminal MDD nodes labeled with variable x_i have exactly N_i outgoing arcs, labeled 0 through $N_i - 1$; if f is the function represented by the node, we write it as $(x_i, f_{x_i=0}, \ldots, f_{x_i=N_i-1})$. Terminal MDD nodes are labeled from the set $\{0, \ldots, m-1\}$. The definitions for *ordered* and *reduced* MDDs are similar to those for BDDs. As with ROBDDs, given a total ordering on the variables, reduced ordered MDDs (ROMDDs) are a canonical representation. We use "MDD" to mean "ROMDD".

An example is depicted in Fig. 4. The MDD shown represents the function $min(a, b, c)$, where a, b, c can take on the values $\{0, 1, 2\}$. The MDD is reduced: no two nodes are equivalent and no node exists with all output arcs equal.

We can manipulate MDDs by using the *Case* operator, defined in [19] by $Case(F, G^0, \ldots, G^{m-1}) = G^i$ if $F = i$, where the range of F is $\{0, \ldots, m-1\}$. A recursive algorithm for computing the *Case* operator based on the relation

$$Case(F, G^0, \ldots, G^{m-1})_{x=i} = Case(F_{x=i}, G^0_{x=i}, \ldots, G^{m-1}_{x=i})$$

is given in Fig. 5. Given reduced MDDs, the algorithm returns a reduced MDD. The reductions are performed in line 10, which ensures that a node with equal arcs is not created, and in line 12, which ensures that two equivalent nodes are not created. Equivalent nodes are detected via a uniqueness table, usually implemented as a hash table. Before a new node is created, the uniqueness table is checked for an equivalent existing node. Another data structure used by an implementation of MDDs is a cache of operations. This prevents duplication of work, as a second call to *Case* with the same parameters will use the result saved in the cache. Both the node uniqueness table and the cache are well-known techniques that apply equally well to MDDs and BDDs [3,19].

Like BDDs, MDDs can be used to represent a set $\mathcal{S}$ of integer tuples by storing the characteristic function $\chi_{\mathcal{S}}$ of the set. Sets can then be manipulated using MDD operations on their characteristic functions. For instance, the union of two sets is computed by $\chi_{\mathcal{A}\cup\mathcal{B}} = \mathit{Union}(\chi_{\mathcal{A}}, \chi_{\mathcal{B}}) = \mathit{Case}(\chi_{\mathcal{A}}, \chi_{\mathcal{B}}, 1)$ and the intersection of two sets is computed by $\chi_{\mathcal{A}\cap\mathcal{B}} = \mathit{Intersect}(\chi_{\mathcal{A}}, \chi_{\mathcal{B}}) = \mathit{Case}(\chi_{\mathcal{A}}, 0, \chi_{\mathcal{B}})$. In the remainder of the paper, we use the operators *Union* and *Intersect* for clarity, with the understanding that they are implemented using the *Case* operator. Also, we will sometimes write $\mathcal{S}$ instead of $\chi_{\mathcal{S}}$, with the understanding that $\mathcal{S}$ is always represented by its characteristic function $\chi_{\mathcal{S}}$.

3 Our Technique

As in any structured approach, we assume that the partition of $\mathcal{P}$ into K sets $\mathcal{P}_K, \ldots, \mathcal{P}_1$ has been performed according to some criterion. In our case, we require a "product-form" decomposition [9], that is:

1. There exist K functions $\mathit{Enabled}_k : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}_k|} \to \{0,1\}$ such that

$$\forall t \in \mathcal{T},\ \mathit{Enabled}(t, \mathbf{m}) \ \Leftrightarrow\ \mathit{Enabled}_K(t, \mathbf{m}_K) \wedge \cdots \wedge \mathit{Enabled}_1(t, \mathbf{m}_1).$$

2. There exist K functions $\mathit{New}_k : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}_k|} \to \mathbb{N}^{|\mathcal{P}_k|}$ such that

$$\forall t \in \mathcal{T},\ \mathbf{n} = \mathit{New}(t, \mathbf{m}) \ \Leftrightarrow\ \mathbf{n}_K = \mathit{New}_K(t, \mathbf{m}_K) \wedge \cdots \wedge \mathbf{n}_1 = \mathit{New}_1(t, \mathbf{m}_1).$$

Such a partition can be found automatically, from a simple inspection of the marking-dependent expressions for I, O, H, and g. Assuming that each $g(t,\cdot)$ is expressed as a conjunction of terms, $g(t,\cdot) = f_1(t,\cdot) \wedge \cdots \wedge f_{r_t}(t,\cdot)$, and letting $\mathcal{G}$ be the union of these terms, over all transitions, algorithm *Partition* can be used for this purpose (Fig. 6). In particular, if I, O, and H are not marking-dependent and g is identically equal 1, the previous two criteria are satisfied by the finest partition, where $K = |\mathcal{P}|$, i.e., each place is in a class by itself. Of course, any coarsening of a product-form partition is itself a product-form partition, so it could be used as well. We illustrate the effect of this choice in Sect. 5, but finding a "good" partition is still an open problem.

A concept of locality for both the enabling condition and the effect of firing, more refined than the one we introduced in [8], is expressed by making use of the functions just introduced. A transition t is local to exactly $\{\mathcal{P}_{k_1}, \ldots, \mathcal{P}_{k_n}\}$ if

$$\forall k \notin \{k_1, \ldots, k_n\},\ \forall \mathbf{m}_k \in \mathcal{S}_k,\ \mathit{Enabled}_k(t, \mathbf{m}_k) = 1 \ \wedge\ \mathit{New}_k(t, \mathbf{m}_k) = \mathbf{m}_k$$

```
Partition(P, T, I, O, H, g)
 1. R = {{p_1}, {p_2}, ..., {p_|P|}};                    • finest possible partition
 2. for each p ∈ P do
 3.     find P_i ∈ R such that p ∈ P_i;
 4.     for each arc with marking dependent cardinality f connected to p and
            each guard expression f ∈ G containing p do
 5.         for each p' ∈ P \ P_i appearing in the expression for f do
 6.             find P_j ∈ R such that p' ∈ P_j;
 7.             R ← R \ {P_i, P_j} ∪ {P_i ∪ P_j};          • merge P_i and P_j
 8. return R;
```

Fig. 6. Algorithm to find the finest product-form partition $\mathcal{R} = \{\mathcal{P}_1, \ldots, \mathcal{P}_K\}$ of $\mathcal{P}$.

(i.e., if $k \notin \{k_1, \ldots, k_n\}$, the marking of $\mathcal{S}_k$ does not affect the enabling, nor is it affected by the firing, of t).

We are now ready to discuss state space generation using MDDs. Since $\mathcal{P}$ has been partitioned into K sets, we use a K-variable MDD to store $\mathcal{S}$. As discussed in Sect. 2, a submarking $\mathbf{m}_k \in \mathcal{S}_k \subset \mathbb{N}^{|\mathcal{P}_k|}$ can be indexed by a (small) integer $m_k \in \{0, 1, \ldots, |\mathcal{S}_k|-1\}$. In the following, and in our implementation, we use this encoding for both simplicity and efficiency. Henceforth, we write $Enabled_k(t, m_k)$ and $n_k = New_k(t, m_k)$ instead of $Enabled_k(t, \mathbf{m}_k)$ and $\mathbf{n}_k = New_k(t, \mathbf{m}_k)$, to stress that we are operating on (local) indices, but we will keep talking about (sub)markings, with the understanding that only indices are really stored in our MDDs.

To generate the reachability set $\mathcal{S}$, we must manipulate the MDD representation of $\mathcal{S}$ to simulate the firing of transitions. We first show how to do this for *local* transitions (local to exactly one $\mathcal{P}_k$), and then for *synchronizing* transitions (local to more than one set of places).

3.1 Local Transition Firing

A transition t local to exactly one set of places $\mathcal{P}_k$ has the special property that t only affects places in $\mathcal{P}_k$. More formally, this says that $Enabled(t, \mathbf{m}) = Enabled_k(t, m_k)$ and $New(t, \mathbf{m}) = [m_K, \ldots, m_{k+1}, New_k(t, m_k), m_{k-1}, \ldots, m_1]$. This implies that, for any reachable marking $[\alpha, m_k, \beta]$ and any local transition t such that $Enabled_k(t, m_k)$, marking $[\alpha, New_k(t, m_k), \beta]$ is also reachable. Thus, the markings $[\alpha, New_k(t, m_k), \beta]$ must be added to $\mathcal{S}$, through the arc update

$$f_{x_k = New_k(t,m_k)} \leftarrow Union(f_{x_k = m_k}, f_{x_k = New_k(t,m_k)}), \tag{1}$$

for each MDD node f in $\chi_{\mathcal{S}}$ labeled with variable x_k. After performing this operation, for any reachable marking $[\alpha, m_k, \beta]$, the marking $[\alpha, New_k(t, m_k), \beta]$ is now reachable. To see this, consider the path in $\chi_{\mathcal{S}}$ for $[\alpha, m_k, \beta]$ ending at terminal node 1. If the path contains a node labeled with x_k, then the update ensures that there is also a path for $[\alpha, New_k(t, m_k), \beta]$ ending at 1, found by following the downward arc $New_k(t, m_k)$ instead of m_k at node x_k. Otherwise,

```
DoLocals(S)
 1. if S = 0 return 0;
 2. if S = 1 return 1;
 3. if cache contains the entry for S then
 4.     return cache entry result;
 5. let x_k be the top variable of S;
 6. Changed ← ∅;
 7. for i ← 0 to N_k − 1 do
 8.     H^i ← DoLocals(S_{x_k=i});
 9.     if H^i ≠ 0 then Changed ← Changed ∪ {i};
10. while Changed ≠ ∅ do
11.     remove some element i from Changed;
12.     for each transition t local to P_k do
13.         if Enabled_k(t, i) then
14.             j ← New_k(t, i);
15.             F ← Union(H^i, H^j);                    • Application of Eq. 1
16.             if F ≠ H^j then
17.                 Changed ← Changed ∪ {j};
18.                 H^j ← F;
19. if H^0 = H^1 = ··· = H^{p−1} then
20.     R ← H^0;
21. else
22.     R ← UniqueTableInsert(x_k, H^0, ..., H^{p−1});
23. add [S, R] to cache;
24. return R;
```

Fig. 7. Algorithm for firing local transitions

the path does not contain a node labeled with x_k, and thus does not depend on variable x_k; in this case, the path for $[\alpha, x_k, \beta]$ ends at 1 for any value of x_k.

The operation performed by Eq. 1 adds to $\mathcal{S}$ all the markings reached by firing a single local transition t when the submarking for $\mathcal{S}_k$ is m_k. To completely simulate the firing of transition t, we perform this operation for all submarkings m_k such that $Enabled_k(t, m_k)$. This can be done "in parallel": each MDD node in $\chi_{\mathcal{S}}$ labeled with variable x_k is visited once, and Eq. 1 is applied to multiple submarkings m_k. In fact, we can perform Eq. 1 for all transitions local to $\mathcal{P}_k$ in one operation.

This idea is the basis for our local transition manipulation algorithm, shown in Fig. 7. Given a set of markings $\mathcal{S}$, *DoLocals* returns the set of markings that can be reached by a marking in $\mathcal{S}$ firing any sequence of local transitions (including none). The algorithm visits each MDD node in $\chi_{\mathcal{S}}$ and, based on the variable label x_k of the node, fires transitions local to $\mathcal{P}_k$ using Eq. 1. Arcs that change are added to the set *Changed*, and are explored again.

3.2 Synchronizing Transition Firing

A transition t local to more than one $\mathcal{S}_k$ requires to check more than one set of places to determine its enabling and the markings reached when it fires. If t is local to $\mathcal{P}_{k_1}, \ldots, \mathcal{P}_{k_n}$, then $Enabled(t, \mathbf{m}) = Enabled_{k_1}(t, m_{k_1}) \wedge \cdots \wedge Enabled_{k_n}(t, m_{k_n})$. We cannot simulate the firing of a synchronizing transition t

by examining MDD nodes in isolation as we did with local transitions. Instead, to add the markings reached when t fires, we must perform three distinct operations: determine the set of markings that enable t, determine the new markings reached after t fires, and add the new markings to the set of reachable markings.

Given t, we compute $\mathcal{E}(t)$, the set of potential markings enabling t, as

$$\chi_{\mathcal{E}(t)}(x_K, \ldots, x_1) = \mathit{Enabled}_K(t, x_K) \wedge \cdots \wedge \mathit{Enabled}_1(t, x_1).$$

Assuming the sets $\mathcal{S}_k$ can be generated *a priori*, the sets $\mathcal{E}(t)$ can also be computed once *a priori* and used throughout the entire state space generation process. Otherwise, $\mathcal{E}(t)$ must be recomputed whenever a new marking is added to $\mathcal{S}_k$. Either way, once we have obtained $\mathcal{E}(t)$, we can compute the set of markings in $\mathcal{S}$ that enable t as $\mathit{Intersect}(\mathcal{S}, \mathcal{E}(t))$.

Then, we simulate the firing of t by updating the submarkings affected by t. For each marking $[m_K, \ldots, m_1]$ that enables t, the marking $[m'_K, \ldots, m'_1]$ is reached after t fires, where $m'_k = \mathit{New}_k(t, m_k)$ if t is local to $\mathcal{P}_k$, $m'_k = m_k$ otherwise. This set of markings is computed using the *Fire* operator, whose algorithm is shown in Fig. 8. Given a set of markings that enable t, *Fire* returns the set of markings reached after t fires by copying the downward arc of m_k from the input set to $\mathit{New}_k(t, m_k)$ in the output set (line 12). Recall that $\mathit{New}_k(t, m_k) = m_k$ for all m_k if t is not local to $\mathcal{P}_k$; thus, *Fire* performs a simple copy in this case. The *Union* operator in line 12 is required because, with marking-dependent arc cardinalities, the firing of t in multiple submarkings i might lead to the same submarking j. Note that the *Fire* operator requires us to specify the parameter x_k, which represents the "current variable", because it may change "don't care" variables. This can occur if every submarking enables t but not every submarking is reached by firing t. Finally, *Fire* returns $\mathcal{S}$ unchanged if the top variable x_k is past $\mathit{Last}(t) = \min\{k : t \text{ is local to } \mathcal{P}_k\}$, the last submarking affected by t.

3.3 Our Generation Algorithm

The generation algorithm is shown in Fig. 9. It consists of two phases per iteration. The first phase, represented by line 3, finds all submarkings that can be reached from the current $\mathcal{S}$ by firing only sequences of local transitions. The second phase, represented by lines 5 through 9, handles the synchronizing transitions.

Thus, the number of iterations required by *MDDexplore* is bounded by one (to recognize that $\mathcal{S}$ has not changed) plus the *synchronizing depth* of the net (as opposed to the sequential depth of [17]) defined as $\max_{\mathbf{m} \in \mathcal{S}}\{d(\mathbf{m})\}$, where

$$\begin{aligned} d(\mathbf{m}) = \min\{n \in \mathbb{N} : \exists\ & (t_1^1, \ldots, t_{l_1}^1, y^1, t_1^2, \ldots, t_{l_2}^2, y^2, \ldots, y^n, t_1^{n+1}, \ldots, t_{l_{n+1}}^{n+1}) \in \mathcal{T}^*, \\ & \text{with } l_i \geq 0,\ t_j^i \text{ local transitions and } y^i \text{ synchronizing transitions,} \\ & \text{whose firing leads from } \mathbf{m}^{[0]} \text{ to } \mathbf{m}\}. \end{aligned}$$

The actual number of iterations can be smaller, depending on the order in which synchronizing transitions are processed: transition y^2 in the sequence

```
Fire(x_k, S, t)
 1. if S = 0 return 0;
 2. if k < Last(t) return S;                    • if k < Last(t) then t does not affect S
 3. if cache contains entry for (x_k, S, t) then
 4.     return cache entry result;
 5. let x_{k'} be the top variable of S;
 6. while (t not local to P_k) and (k > k') do
 7.     k ← k − 1;
 8. for i ← 0 to N_k − 1 do
 9.     H^i ← 0
10. for i ← 0 to N_k − 1 do
11.     j ← New_k(t, i);
12.     H^j ← Union(H^j, Fire(x_{k−1}, S_{x_k=i}, t));
13. if H^0 = H^1 = ··· = H^{N_k−1} then
14.     R ← H^0;
15. else
16.     R ← UniqueTableInsert(x_k, H^0, ..., H^{N_k−1});
17. add [(x_k, S, t), R] to cache;
18. return R;
```

Fig. 8. Algorithm for firing synchronizing transitions

```
MDDexplore(m^[0])
1. S ← (x_K = m_K^[0]) ∧ ··· ∧ (x_1 = m_1^[0]);      • Set S to the initial marking
2. repeat forever
3.     S ← DoLocals(S);
4.     O ← S;                                        • Save old set of reachable states
5.     for each synchronizing transition t do
6.         E ← Intersect(S, E(t));                   • E is the set of markings that enable t
7.         F ← Fire(x_K, E, t);                      • F is the set of markings reached after t fires
8.         S ← Union(S, F);                          • Add F to S
9.     if O = S return S;
```

Fig. 9. Algorithm to compute $\mathcal{S}$

(y^1, y^2) leading from $\mathbf{m}^{[0]}$ to $\mathbf{m}$ is considered during the first iteration if y^1 is processed before y^2, during the second iteration otherwise ([18] already recognized the importance of a good "chaining" order).

3.4 Logical Queries on the State Space

Once $\mathcal{S}$ has been generated, it can be used to answer various classes of logical queries, by performing MDD operations. For instance, suppose we want to compute the set of reachable markings satisfying some boolean condition q. To do this, we first build the set $\mathcal{Q}$ of potential markings in $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ that satisfy q, and then compute $\mathcal{S} \cap \mathcal{Q}$. First, let us consider a simple query whose condition is enforced at a single level k. Given a condition $f_k : \mathcal{S}_k \to \{0, 1\}$, we compute the set of submarkings $\mathcal{Q}$ that satisfy f_k:

$$\chi_{\mathcal{Q}}(x_K, \ldots, x_1) = f_k(x_k).$$

Complex queries are answered by combining simple queries. An important example is asking for the set of reachable (absorbing, or dead) markings that do not enable any transition. First, we compute $\mathcal{Q}(t)$, the set of potential markings that do not enable t by

$$\chi_{\mathcal{Q}(t)}(x_K, \ldots, x_1) = (\mathit{Enabled}_K(t, x_K) = 0) \vee \cdots \vee (\mathit{Enabled}_1(t, x_1) = 0)$$

for all transitions t. Next, we compute the set of potential markings that do not enable any transition, $\mathcal{Q} = \bigcap_{\forall t \in \mathcal{T}} \mathcal{Q}(t)$. Finally, $\mathcal{S} \cap \mathcal{Q}$ gives us the reachable absorbing markings.

Another important class of queries deals with the possibility of a condition b occurring after a condition a has occurred. To answer this question we build the subsets $\mathcal{Q}_a$ and $\mathcal{Q}_b$ of $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ satisfying a or b, respectively. Then, we run *MDDexplore* except that, in line 1, $\mathcal{S}$ is initialized to the set $\mathcal{S} \cap \mathcal{Q}_a$ of reachable markings satisfying a, instead of the initial marking. The intersection of the resulting $\mathcal{S}$ with $\mathcal{Q}_b$ gives exactly the set of markings satisfying b that can be reached from some reachable marking satisfying a. In the worst case, this approach requires as many iterations as for the original generation of $\mathcal{S}$.

4 Implementation Issues

One way to implement a MDD data structure is to map each MDD node onto a BDD structure using some encoding. With this approach, MDD operators are translated into equivalent operations on the underlying BDD. Indeed, both Kam's PhD thesis [11] and the timing results in [19] seem to suggest that this achieves high(er) efficiency. Instead, we choose to implement MDDs directly, where each MDD node contains some node information (the variable index, the number of downward pointers, etc.) and an array of downward pointers. As the next section shows experimentally, our approach can be vastly superior (at least for the examples we used). In the the first two applications, assigning one (safe) place per level effectively reduces our implementation to the BDD case, and doing so results in much higher execution times (even if locality is exploited in either case) and memory consumption.

To conserve memory in our implementation, MDD nodes store node indices instead of full pointers. In particular, MDD nodes labeled with variable x_1 can only have downward pointers to terminal nodes 0 and 1; thus, an array of bits is used. More sophisticated MDD node storage schemes would be possible, including sparse array storage and other forms of compression. This is because we do not directly modify MDD nodes: we use temporary full integer arrays during MDD node construction (e.g., for storing $H^0, \ldots, H^{N_k-1}$ in algorithm *Case*), which are copied into appropriately-sized structures for long-term storage. In our experience, this compression introduces a 10%-20% CPU overhead, but it typically reduces memory usage by a factor of 4.

In our studies, we have assumed that the local reachability sets $\mathcal{S}_k$ (or supersets of them) are computable in isolation, i.e., before the generation of the overall reachability set $\mathcal{S}$. This is a common assumption with structured approaches

(e.g., [4,13]), and certainly with safe nets, for which $\mathcal{S}_k \subseteq \{0,1\}^{|\mathcal{P}_k|}$ by definition. This restriction can be lifted by dynamically generating local reachability sets $\mathcal{S}_k$ during generation of $\mathcal{S}$. This requires a more complex implementation of $Enabled_k$ and New_k, where the first time $New_k(t, \mathbf{m}_k)$ is called on $\mathbf{m}_k$, we compute the reached submarking $\mathbf{n}_k$ and add it to $\mathcal{S}_k$, if necessary.

We conclude this section with an observation. At each iteration, algorithm *MDDexplore* considers the firing of sequences of local transitions before examining the synchronizing transition. This is in contrast to the algorithm proposed in [17], which does not exploit the concept of locality, and is one of the reasons for our greater efficiency. As exploring the firing for local transitions is considerably less expensive than for the synchronizing transitions, we achieve two goals: we reduce the number of iterations required by *MDDexplore* (compare the sequential depth of [17] with our synchronizing depth) and we add more markings to $\mathcal{S}$ every time we explore the firing of a synchronizing transition (since we fire it from a set $\mathcal{S}$ already augmented by the local firings).

5 Results

We now apply our approach to various models taken from the literature. We examine first the dining philosophers and the slotted ring models presented in [17]. These are safe Petri nets composed of several identical subnets: the state space size is increased by adding subnets. Then, we consider the flexible manufacturing and Kanban systems presented in [8]. These models have a fixed number of places and transitions: the state space size increases with the initial marking.

We implemented our approach in the tool SMART [7]. Our results are obtained on a 400 MHz Pentium II workstation under the Linux operating system. No run made use of virtual memory. For each experiment, Table 1 reports:

- The size of the state space $\mathcal{S}$.
- The final and peak number of MDD nodes (our data structure grows and contracts during the execution of *MDDexplore*).
- The final and peak memory consumption, in bytes (peak memory is an indicator of the amount of RAM required to avoid relying on virtual memory, while final memory is of interest in case $\mathcal{S}$ is saved for further use).
- The number of iterations performed by *MDDexplore*.
- The overall CPU times, in seconds.

We also explore the effect of different partitioning of the model into levels, in Fig. 11. Occasional "bumps" in the plots are artifacts of the compression used in our implementation.

5.1 Dining Philosophers

The dining philosopher model is composed of N subnets. The Petri net for the i^{th} philosopher is shown in Fig. 10(a). The net represents a philosopher and the philosopher's right fork. The philosopher's left fork, represented by the dotted

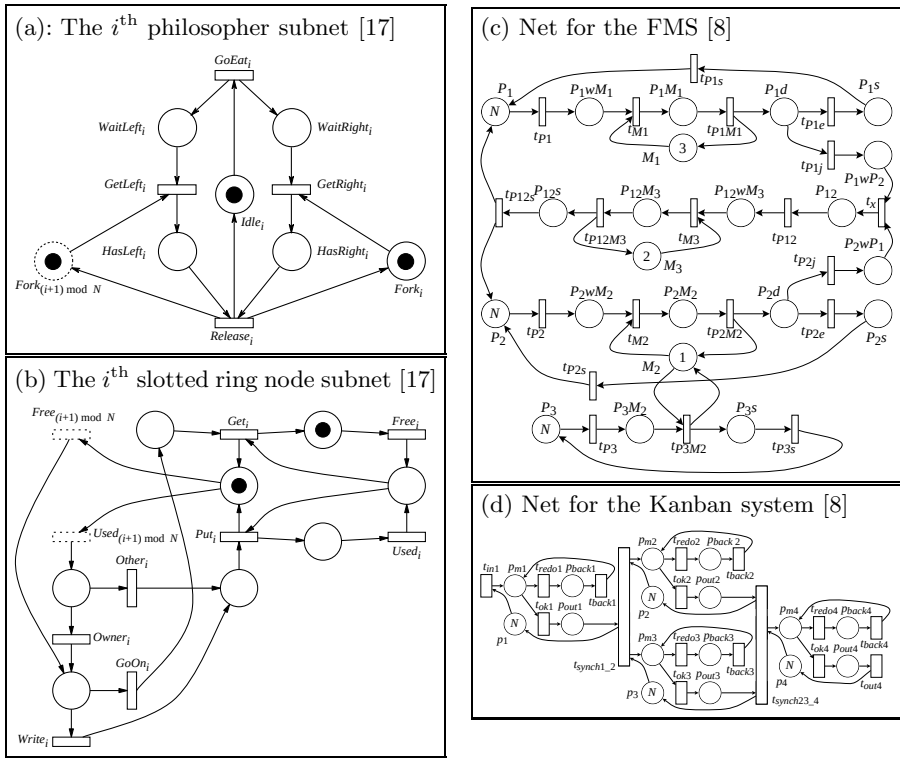


Fig. 10. The four models used in our experiments.

place $Fork_{(i+1) \bmod N}$, is part of the subnet for the next philosopher; it is depicted to illustrate how the subnets interact.

A natural partitioning scheme is to assign each philosopher to a separate level. We also investigate grouping two or three adjacent philosophers together in the same level. Table 1 shows the results for two philosophers per level. We find that grouping more than two philosophers in each level results in worse memory usage and CPU times (Fig. 11). However, there is an interesting tradeoff between having one or two philosophers per level: the former choice results in higher execution times, but lower memory requirements. The grouping where each place is in a different level corresponds to the BDD approach in [17], and it is vastly less efficient.

"True" MDDs with locality are extremely efficient for this example, both in terms of memory usage and generation time. There are two main reasons for this. First, our approach requires only two iterations, no matter how many philosopher subnets are present. The synchronizing depth of the model grows as N, since the synchronizing transitions are $GetLeft_i$, $GetRight_i$, and $Release_i$, and the farthest markings are those where N forks are taken, as left or right forks. However, any such marking $\mathbf{m}$ can be reached from $\mathbf{m}^{[0]}$ through N!

different firing sequences (if we ignore the position of local transition $GoEat_i$ in the sequence), and there is always one sequence that respects the order in which the synchronizing transitions are considered in *MDDexplore*. In other words, the entire reachability set has been discovered by the second execution of statement 3 in *MDDexplore*. In contrast, the number of iterations reported for the BDD approach grows as $2N + 1$ [17].

Second, the MDD contains exactly four distinct nodes at each level (except for the top level, which always has one), no matter how many philosophers we add or how many (adjacent) philosophers we group into each level. This is because a philosopher will either hold both forks, only the left fork, only the right fork, or no fork at all. This is still the case for a level containing several adjacent philosophers, where the left and right forks are the boundary forks between levels. Hence, the final number of MDD nodes in Table 1 is $4(N/2 - 1) + 1 = 2N - 3$. It is interesting to note that the peak number of nodes is also linear, $6N - 15$.

5.2 Slotted Ring Network

The Petri net for a single node of a slotted ring network protocol is shown in Fig. 10(b). The overall model is composed of N such subnets connected by sharing transitions ($Free_{(i+1) \bmod N}$ and $Used_{(i+1) \bmod N}$). Table 1 shows the results for a decomposition where each node is in a different level. The effect of other choices, one place per level (essentially a BDD) and two nodes per level, is as for the previous model (Fig. 11). The number of iterations required by *MDDexplore* is $N/2 + 2$, while that for the BDD approach in [17] grows quadratically.

5.3 Flexible Manufacturing System

The FMS model shown in Fig. 10(c) [8] is parameterized by the initial number N of tokens in P_1, P_2, and P_3. We compare three different partitioning schemes in Fig. 11. The model is partitioned into 4, 6, or 19 levels. In the first case, the partition is $\{P_1,P_1wM_1,P_1M_1,M_1,P_1d,P_1s,P_1wP_2\}$, $\{P_{12},P_{12}wM_3,P_{12}M_3,M_3,P_{12}s\}$, $\{P_2,P_2wM_2,P_2M_2,M_2,P_2d,P_2s,P_2wP_1\}$, and $\{P_3,P_2M_2,P_3s\}$. In the second case, it is $\{P_{12},P_{12}wM_3,P_{12}M_3,M_3,P_{12}s\}$, $\{P_1,P_1wM_1,P_1M_1,M_1\}$, $\{P_1d,P_1s,P_1wP_2\}$, $\{P_2,P_2wM_2,P_2M_2,M_2\}$, $\{P_2d,P_2s,P_2wP_1\}$,and $\{P_3,P_2M_2,P_3s\}$. Finally, the partition with 19 levels is obtained by assigning each place to a different level, with the exception of the complementary places M_1, M_2, and M_3, placed in the same level as the places P_1M_1, P_2M_2, and $P_{12}M_3$, respectively.

In this model, the effect of the partition choice is extremely noticeable in terms of both memory and execution times, and the finest partition is by far the best. Table 1 reports the detailed results for the 19-level partition. In this case, *MDDexplore* requires $N + 5$ iterations ($N + 1$ and $N + 2$ are instead required with the 4-level and 6-level partitions, respectively).

5.4 Kanban System

Fig. 10(d) shows the Petri net of a Kanban system [8]. This model is parameterized by the number of tokens N initially in p_1, p_2, p_3, and p_4. Also for this

	N	$\|\mathcal{S}\|$	MDD Nodes Final	MDD Nodes Peak	Memory (bytes) Final	Memory (bytes) Peak	# of Iters.	CPU (secs)
Philosophers	10	1.86×10^{6}	17	45	1,046	2,934	2	0.14
(2 phils/level)	20	3.46×10^{12}	37	105	2,446	7,134	2	0.37
	50	2.23×10^{31}	97	285	6,646	21,498	2	1.61
	100	4.97×10^{62}	197	585	15,788	58,164	2	5.67
	200	2.47×10^{125}	397	1,185	41,044	128,556	2	21.73
	300	1.23×10^{188}	597	1,785	65,124	197,604	2	47.12
	400	6.10×10^{250}	797	2,385	87,608	265,308	2	89.66
	500	3.03×10^{313}	997	2,985	110,008	332,508	2	134.43
	600	1.51×10^{376}	1,197	3,585	133,080	400,380	2	205.44
	700	7.48×10^{438}	1,397	4,185	155,648	467,748	2	296.31
	800	3.72×10^{501}	1,597	4,785	177,796	534,612	2	404.06
	900	1.85×10^{564}	1,797	5,385	200,280	601,980	2	498.25
	1000	9.18×10^{626}	1,997	5,985	222,764	669,180	2	630.40
Slotted Ring	10	8.29×10^{9}	60	691	3,186	38,527	7	1.80
(1 node/level)	20	2.73×10^{20}	220	4,546	12,601	263,002	12	41.13
	30	1.04×10^{31}	480	15,101	27,741	875,327	17	297.97
	40	4.16×10^{41}	840	37,066	48,606	2,149,372	22	1309.66
	50	1.72×10^{52}	1,300	76,308	85,486	5,342,493	27	4282.80
	60	7.29×10^{62}	1,860	139,852	149,106	11,902,318	32	11948.90
FMS	5	2.90×10^{6}	149	433	7,214	21,190	10	0.62
(19 levels)	10	2.50×10^{9}	354	1,038	22,210	65,826	15	2.65
	15	2.17×10^{11}	634	1,868	48,600	144,656	20	7.03
	20	6.03×10^{12}	989	2,923	89,411	266,707	25	14.77
	25	8.54×10^{13}	1,419	4,203	147,632	440,968	30	29.75
	50	4.24×10^{17}	4,694	13,978	804,785	2,410,321	55	260.02
	75	6.98×10^{19}	9,844	29,378	2,347,088	7,034,824	80	1047.90
	100	2.70×10^{21}	16,869	50,403	5,149,541	15,439,477	105	2845.66
Kanban	5	2.55×10^{6}	7	47	294	3,458	11	0.15
(4 levels)	10	1.01×10^{9}	12	87	1,018	22,318	21	1.83
	15	4.70×10^{10}	17	127	2,924	85,768	31	9.72
	20	8.05×10^{11}	22	167	7,049	238,473	41	33.96
	25	7.68×10^{12}	27	207	14,692	540,628	51	92.69
	50	1.04×10^{16}	52	407	197,687	9,510,714	101	2383.10
	75	7.83×10^{17}	77	607	877,068	51,881,884	151	17715.40

Table 1. Results for our models.

model we compare different partitioning schemes, in Fig. 11, corresponding to either 4 or 16 levels. The former case is as indicated by the subscripts 1, 2, 3, and 4 in Fig. 10(d), while the latter case assigns one place to each level. In this case, unlike the FMS, the finer partition is much worse than the coarser one. Table 1 shows the results for the 4-level partition. In this case, *MDDexplore* requires $2N + 1$ iterations, while the 16-level partition requires 5 iterations for $N = 1$ and $3N + 1$ iterations for $N > 1$.

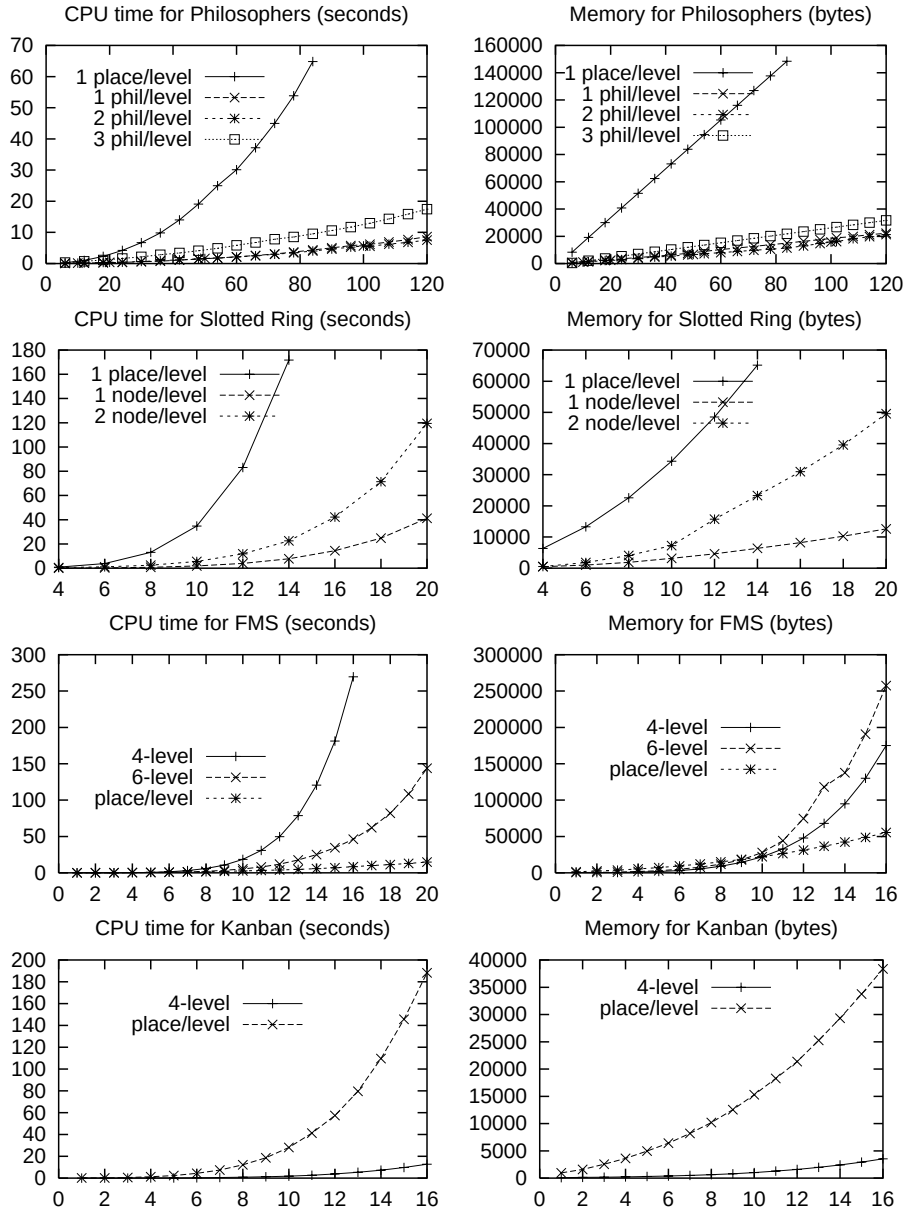


Fig. 11. Effect of different partitions on our models (N on the horizontal axis).

6 Conclusion

We presented a new technique for the generation and storage of the reachability set of a Petri net, closely related to recently proposed BDD-based approaches. However, our use of multi-valued (not boolean) sets and the exploitation of locality to reduce both the number of iterations and the cost of each iteration in the generation procedure result in the ability to tackle much larger reachability sets than previously possible.

The application of our results goes beyond that of Petri net analysis, as it widens the size of the discrete-state systems for which an exhaustive logical verification might be reasonably attempted.

Much work remains to be done, however. We have demonstrated how the choice for the partition of the Petri net places (i.e., the decomposition of the discrete-state model), the order in which the levels are considered, and the order in which the synchronizing transitions are processed can have a substantial effect on the memory and time requirements of the approach. The existence of an efficient algorithm that can derive an optimal strategy for these choices is unlikely, but we shall seek heuristics that work well on models of practical interest.

References

1. S. C. Allmaier, M. Kowarschik, and G. Horton. State space construction and steady-state solution of GSPNs on a shared-memory multiprocessor. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 112–121, St. Malo, France, June 1997. IEEE Comp. Soc. Press.
2. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, Aug. 1986.
3. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comp. Surv.*, 24(3):393–318, 1992.
4. P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 81–90, St. Malo, France, June 1997. IEEE Comp. Soc. Press.
5. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. ICASE Report 97-66 (NASA/CR-97-206274), Institute for Computer Applications in Science and Engineering, Hampton, VA, 1997. Submitted for publication.
6. G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain)*, pages 179–198. Springer-Verlag, June 1994.
7. G. Ciardo and A. S. Miner. SMART: Simulation and Markovian Analyzer for Reliability and Timing. In *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS'96)*, page 60, Urbana-Champaign, IL, USA, Sept. 1996. IEEE Comp. Soc. Press.
8. G. Ciardo and A. S. Miner. Storage alternatives for large structured state spaces. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 1245, pages 44–57, St. Malo, France, June 1997. Springer-Verlag.

9. G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996.
10. S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets)*, pages 258–277, Zaragoza, Spain, June 1994. Springer-Verlag.
11. T. Kam. *State Minimization of Finite State Machines using Implicit Techniques.* PhD thesis, University of California at Berkeley, 1995.
12. P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. Softw. Eng.*, 22(4):615–628, Sept. 1996.
13. P. Kemper. Reachability analysis based on structured representations. In J. Billington and W. Reisig, editors, *Application and Theory of Petri Nets 1996, Lecture Notes in Computer Science 1091 (Proc. 17th Int. Conf. on Applications and Theory of Petri Nets, Osaka, Japan)*, pages 269–288. Springer-Verlag, June 1996.
14. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Syst. Techn. J.*, 38(4):985–999, July 1959.
15. E. Pastor and J. Cortadella. Efficient encoding schemes for symbolic analysis of Petri nets. In *Proc. Design Automation and Test in Europe*, Feb. 1998.
16. E. Pastor and J. Cortadella. Structural methods applied to the symbolic analysis of Petri nets. In *Proc. IEEE/ACM International Workshop on Logic Synthesis*, June 1998.
17. E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain)*, pages 416–435. Springer-Verlag, June 1994.
18. O. Roig, J. Cortadella, and E. Pastor. Verification of asynchronous circuits by BDD-based model checking of Petri nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995, Lecture Notes in Computer Science 935 (Proc. 16th Int. Conf. on Applications and Theory of Petri Nets, Turin, Italy)*, pages 374–391. Springer-Verlag, June 1995.
19. A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *International Conference on CAD*, pages 92–95. IEEE Computer Society, 1990.
20. D. Zampunièris. *The Sharing Tree Data Structure, Theory and Applications in Formal Verification.* PhD thesis, Department of Computer Science, University of Namur, Belgium, 1997.

Structural Methods to Improve the Symbolic Analysis of Petri Nets

Enric Pastor[1], Jordi Cortadella[2], and Marco A. Peña[1]

[1] Department of Computer Architecture
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
enric,marcoa@ac.upc.es
http://www.ac.upc.es/recerca/VLSI/
[2] Department of Software
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
jordic@lsi.upc.es

Abstract. Symbolic techniques based on BDDs (Binary Decision Diagrams) have emerged as an efficient strategy for the analysis of Petri nets. The existing techniques for the symbolic encoding of each marking use a fixed set of variables per place, leading to encoding schemes with very low density. This drawback has been previously mitigated by using Zero-Suppressed BDDs, that provide a typical reduction of BDD sizes by a factor of two.
Structural Petri net theory provides P-invariants that help to derive more efficient encoding schemes for the BDD representations of markings. P-invariants also provide a mechanism to identify conservative upper bounds for the reachable markings. The unreachable markings determined by the upper bound can be used to alleviate both the calculation of the exact reachability set and the scrutiny of properties. Such approach allows to drastically decrease the number of variables for marking encoding and reduce memory and CPU requirements significantly.

1 Introduction

Petri nets (PNs) are a graph-based mathematical formalism that allows to describe systems modeling causality, concurrency and conflict relations among its events [16, 7]. In particular, PNs play an important role in the synthesis and verification of concurrent systems. PNs are applied, for example, to the synthesis and verification of digital asynchronous circuits, to model heterogeneous systems in hardware/software codesign frameworks, and to verify concurrent systems [5, 19].

Symbolic analysis of PNs suffer from the state explosion problem [18, 19]. The number of reachable markings grows exponentially with the size of the PN description. Temporal logic analysis, hazard verification or circuit synthesis, need to express conditions in terms of sets of markings or sequences of events.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 26–45, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Therefore, the size of the representation of the overall state space is critical and limits the efficiency of formal methods on large practical examples. The major goal of the ongoing research on symbolic analysis of PNs is to increase the size of the systems that can be analyzed.

Along the last decades PNs have been deeply investigated with a large number of theoretical results available in the literature. Specially, structural theory connects the dynamic behavior of PNs with its underlying structure [10, 15, 13]. Until recently none of these well-known results has been used in order to alleviate the BDD-based symbolic analysis of PNs.

This work discusses several techniques for the symbolic analysis of PNs. We will show how structural and symbolic techniques can be efficiently combined in the same framework. Sets of P-invariants that will be retrieved from the PN can be applied to ease the symbolic analysis. Previous analysis techniques already make use of binary vector representations of markings [11] and P-invariants to reduce the number of bits in the vector representations [6]. However, they did not exploit the fact that the combination of P-invariants and BDDs already provide information about the reachable markings in the PN.

The proposed algorithms can be classified in two groups according to their application to the computation and representation of the reachability set.

- The first set improves the symbolic BDD representation of the reachability set, reducing the number of required Boolean variables and BDD nodes. Encoding algorithms will be proposed both for the subclass of safe PNs and for general bounded PNs.
- The second set provides enlarged approximations of the reachability set that can be efficiently computed. This approximations can be applied to conservative verification techniques or to provide approximations of the sets of unreachable markings to further reduce the BDD representations.

The outline of the paper is the following. In Section 2 we introduce some basic notions on PNs and Boolean functions. Section 3 describes by means of an example how symbolic BDD techniques currently encode PNs and demonstrates the existence of room for further improvement. An algorithm to encode safe PNs is presented in Section 4. The algorithm is based on sets of one-token SM-Components, assigning to each place in an SM-Component a unique Boolean function. Section 5 extends the encoding methodology to any bounded PN by using general P-invariants. Each potential token configuration in a P-invariant is assigned a unique Boolean function. Additionally, P-invariants allow to determine a set of Potentially Reachable Markings. In Section 6 we show that computing a conservative set of unreachable markings may help to further improve the analysis of PNs. Finally, Section 7 presents experiments that demonstrate the efficiency of the proposed encoding techniques. Section 8 concludes the paper and introduces some future research directions.

2 Basic Notations

2.1 Petri Nets

A *Petri net* (PN) [16, 7] is a four-tuple $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}, M_0 \rangle$, where $\mathcal{P}$ and $\mathcal{T}$ are sets of places and transitions respectively. $\mathcal{W} : (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}) \rightarrow \mathbb{N}$ defines the *weighted flow relation*. If $\mathcal{W}(u, v) > 0$ then there is an arc from u to v with *weight* $\mathcal{W}(u, v)$. The function $M : \mathcal{P} \rightarrow \mathbb{N}$ is called a *marking*; that is, an assignment of a nonnegative integer to each place. If k is assigned to place p in a marking M, we will say that p is marked with k tokens ($M(p) = k$). M_0 is the initial marking of the PN.

PNs are graphically represented by drawing places as circles, transitions as boxes (or sometimes bars), the flow relation as directed arcs, and tokens as dots circumscribed into the places. Fig. 1(a) depicts a PN with initial marking $M_0 = \{M_0(p_1) = 2, M_0(p_2) = 0, M_0(p_3) = 1, M_0(p_4) = 0\}$.

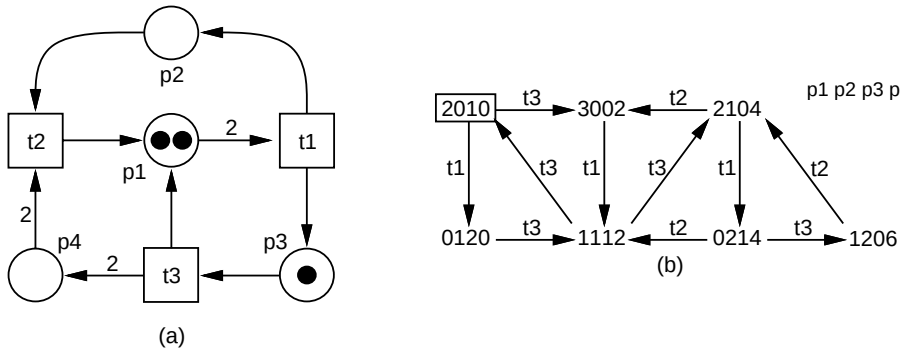


Fig. 1. (a) A bounded PN and (b) its reachability graph.

The set of markings that can be reached from the initial marking M_0 by repeatedly firing the transitions of the net is called the *reachability set* (denoted RS). Fig. 1(b) shows the reachable markings corresponding to the PN example in Fig. 1(a).

A place $p \in \mathcal{P}$ is called *k-bounded* ($k \in \mathbb{N}$) iff at any reachable marking it does not contain more than k tokens. A PN is *bounded* iff every place is k-bounded for some value k. A PN is *safe* if all places are 1-bounded. Fig. 2 depicts a safe PN describing two competing philosophers.

PNs can be symbolically manipulated by means of BDDs [18, 17, 19]. Each place in the PN is considered as an integer variable, being represented by a number of Boolean variables. The RS can be obtained by computing the least fix point of the following recurrence [5]:

$$S_o = M_0$$
$$S_{i+1} = S_i \cup Image(PN, S_i)$$

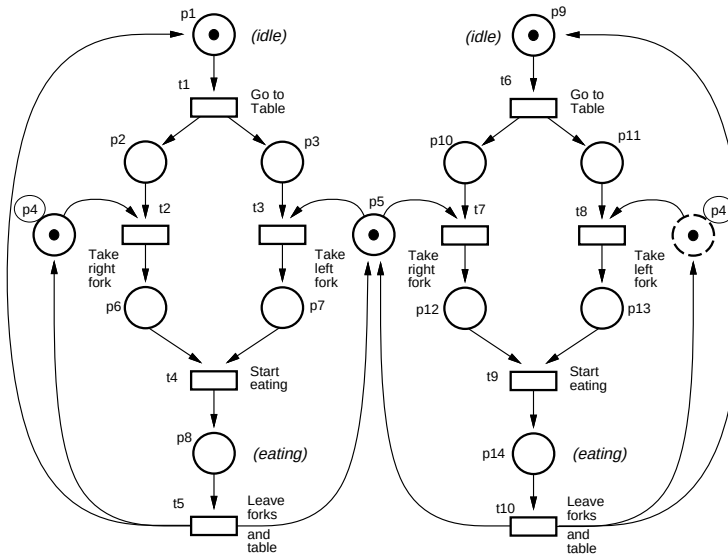


Fig. 2. PN for two dining philosophers (two instances of p_4 are depicted for clarity).

where *Image* is a function that returns the markings reachable from S_i in one step. In the PN example of Fig. 1(a), $\text{Image}(PN, [2010]) = \{[3002], [0120]\}$.

From now on, we will assume that the reader to be familiar with both BDDs and Algebraic Decision Diagrams (ADDs) [2, 4, 1].

2.2 Place-Invariants and State Machines Components

The structure of a PN can be represented by its *incidence matrix* [13, 15], a $\mathcal{P} \times \mathcal{T}$ integer matrix given by $C(p_i, t_j) = \mathcal{W}(t_j, p_i) - \mathcal{W}(p_i, t_j)$. The incidence matrix of the PN depicted in Fig. 1(a) is the following:

$$C = \begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{pmatrix}$$

The places of a PN have an associated token conservation equation usually written in the matrix form $M = M_0 + C \cdot \boldsymbol{\sigma}$, where $\boldsymbol{\sigma}$ is called the *Parikh vector* of a sequence of transitions σ.

Every solution $X \in \mathbb{Q}^{|\mathcal{P}|}$ of the equation $X^T \cdot C = 0$ is said to be a P-invariant [16]. A P-invariant I is called *semi-positive* if $I \geq 0$ and $I \neq 0$. The support of a semi-positive P-invariant I, denoted by $\langle I \rangle$, is the set of places p satisfying $I(p) > 0$. A semi-positive P-invariant I is *minimal* if no other semi-positive P-invariant J satisfies $\langle J \rangle \subset \langle I \rangle$. In the sequel, for sake of simplicity we will refer to P-invariants as *invariants*.

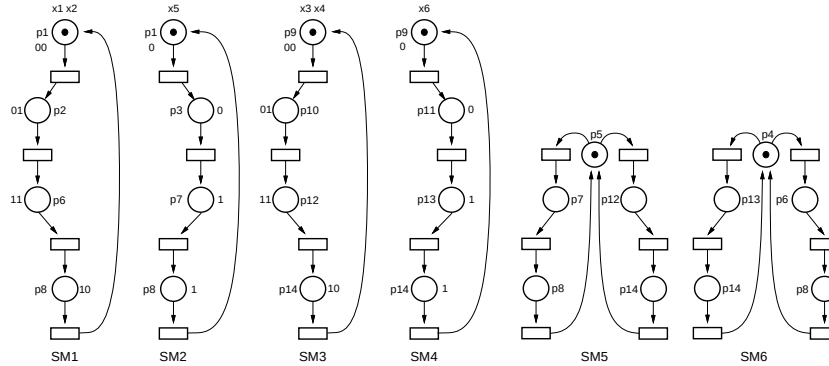


Fig. 3. SM decompositions for the dining philosophers example.

Given an invariant I, any reachable marking M must agree with the initial marking M_0; that is, $I \cdot M_0 = I \cdot M$. Therefore, invariants can be used to prove that a marking M is not reachable if M and M_0 do not agree on an invariant.

The PN N_i generated by a subset of places is said to be a *State Machine Component* (SM) of N if N_i is a strongly connected State Machine. A key result for the contribution of this work is the following [7]: *Let* $N_i = \langle \mathcal{P}_i, \mathcal{T}_i, \mathcal{W}_i, M_{0i} \rangle$ *be a SM-Component of a Petri Net* N*. Then* N_i *is a minimal semi-positive P-invariant of* N*.*

The *Smith Normal Form* [12] provides an efficient method to derive invariants for bounded PNs. This technique has been introduced by Desel et al. [8] and generates a basis of all possible invariants (not necessarily minimal or semi-positive). A basis of invariants for the PN in Fig. 1(a) is:

$$\begin{aligned} I_1 &: 2p_1 + 4p_2 - p_4 = 4 \\ I_2 &: p_1 + p_2 + p_3 = 3 \ . \end{aligned} \tag{1}$$

For safe PNs we can compute SM-Components by posing a set of linear equations [15, 13]. A minimal one-token semi-positive invariant I_{P}, including a place p_i, can be computed by solving the linear system of equations:

$$\begin{aligned} \mathit{min} \quad & \textstyle\sum_p I_{\mathrm{P}}(p) \geq 0 \\ \mathit{s.t.} \quad & I_{\mathrm{P}} \cdot C = 0 \\ & \textstyle\sum_p I_{\mathrm{P}}(p) \cdot M_0(p) = 1 \\ & I_{\mathrm{P}}(p_i) \geq 1 \end{aligned}$$

Figure 3 shows six SM-Components generated from the PN of Fig. 2. For example, SM1 has been generated from the invariant $p_1 + p_2 + p_6 + p_8 = 1$ and SM5 from $p_5 + p_7 + p_8 + p_{12} + p_{14} = 1$.

2.3 Logic Functions

Now we briefly sketch some basic theory on Boolean algebras. Most of the concepts presented here have been extracted from [3].

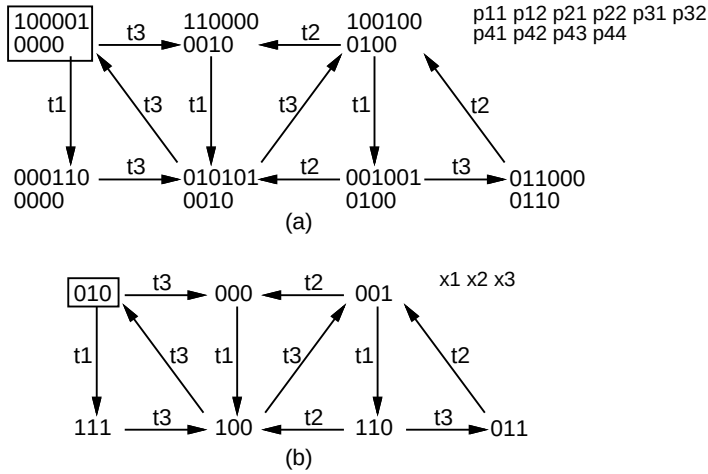


Fig. 4. Encoded reachability graph: (a) sparse and (b) optimal, for the PN in Fig. 1.

A *Boolean algebra* is a quintuple $(\mathrm{B}, +, \cdot, 0, 1)$ where B is a set called the carrier, $+$ and $\cdot$ are binary operations on B, and 0 and 1 are elements of B. The system $(\mathrm{B} = \{0, 1\}, +, \cdot, 0, 1)$, with $+$ and $\cdot$ defined as the *logic OR* and *logic AND* operations respectively, is a Boolean algebra.

An n-variable *logic function* is a mapping $f : \mathrm{B}^n \rightarrow \mathrm{B}$. Let $\mathcal{F}_n$ be the set of n-variable logic functions on B. Then the system $(\mathcal{F}_n, +, \cdot, 0, 1)$ is also a Boolean algebra. Let us also define $\mathcal{M}_n$ as the subset of n-variable logic functions such that one and only one combination of inputs evaluates to 1 (i.e. that only contain a minterm).

3 A Motivating Example

The symbolic representation of the *RS* of a PN requires an encoding mechanism to map each marking in a unique binary code inside a Boolean algebra. Traditionally, the encoding has been created by assigning a number of Boolean variables to each place in the PN. The number of variables should be enough to represent the maximum number of tokens that can be located in each place. This encoding technique is known as an *sparse* encoding scheme [17].

Sparse encoding schemes are extremely inefficient for PNs because they assume that all possible combinations of tokens inside places are possible. However, in most cases, places are causally related or in conflict and therefore not all combinations of tokens exist.

In order to compare the efficiency of different encoding schemes we introduce an *encoding density* function $\mathcal{D}$. Given a PN, $\mathcal{D}_{PN}$ is calculated as the optimum

number of variables required to encode the RS, divided by the actual number of variables that are used, *i.e.*

$$\mathcal{D}_{PN} = \frac{\lceil \log_2(|[M_0\rangle|) \rceil}{\#\text{ of variables}}$$

An encoding is optimal if $\mathcal{D}_{PN} = 1$. This optimality implies that the Boolean space is fully used and no binary code is left unassigned.

The bounded PN in Fig. 1(a) has four different places that may contain a certain number of tokens. The maximum number of tokens that can be located in each place will determine how many variables are required for sparse encoding. A conservative upper bound for each place p_i can be derived from a basis B of invariants, i.e.

$$\begin{array}{ll} max & M(p_i) \\ s.t. & B^t M = B^t M_0 \end{array}$$

For that example, after solving the Linear Programming Problem we obtain that the maximum number of tokens are $\mathsf{max}(p_1) = \mathsf{max}(p_2) = 3$, $\mathsf{max}(p_3) = 2$ and $\mathsf{max}(p_4) = 8$. That implies that to encode places p_1, p_2, p_3 we need 2 variables for each of them (because their values are between 0 and 3); while place p_4 requires 4 variables (because its values are between 0 and 8). This sparse scheme leads to a Boolean algebra with 10 variables, representing up to 2^{10} different markings. However, it is known that the PN has only 7 reachable markings! (see Fig. 4(a).)

An *optimal* encoding should use a logarithmic number of variables with respect to the number of reachable markings ($\lceil \log_2 |[M_0\rangle| \rceil$). In the previous example, $\lceil log_2\ 7 \rceil = 3$ is the optimal number of variables (see Fig. 4(b)).

Deriving optimal encoding schemes is not a viable strategy because it requires knowing the existing markings a priori, which is in fact the problem that was originally posed. Hence, the goal of this work is to propose alternative dense encoding schemes for PNs, that lay in between the conventional sparse encoding and the optimal schemes. The proposed methodology should reduce the number of variables, while maintaining a reasonable computation effort.

It is well known that the number of BDD variables does not always have a direct impact on the number of BDD nodes required to represent a set of markings. For a fixed set and number of variables, the number of BDD nodes may vary from polynomial to exponential depending on the variable ordering in the BDD. However, experiments show that a reduction in the number of variables combined with an accurate assignment of binary codes to markings provide significant improvements both in the number of BDD nodes and their computation times.

Finding out relations among places that restrict their simultaneous marking may help to reduce the number of Boolean variables required to encode the same RS [17]. Relations among places not only provide an encoding mechanism, but additionally restrict the set of potentially reachable markings. Some markings will be determined not to be in the RS even before starting any symbolic traversal.

In this work we will propose encoding schemes based on the information known a priori from the PN structure —its invariants. These invariants allow to discard sets of unreachable markings and find more efficient encodings for those that are still potentially reachable. The method proposed for the dense encoding of the reachable markings of a PNs is structured as:

1. A basis of invariants of the PN is calculated. Algebraic and linear programming techniques will be used for ordinary PNs, while the SNF can be used for bounded PNs.
2. The PN must be bounded and the upper bounds must be known, either derived from the invariants or provided by the user.
3. A dense encoding is derived for the places covered by invariants. The rest of places are encoded by using the sparse scheme. Efficient encoding techniques are used for one-token SM-Components, while more elaborated mechanisms are required for general invariants.
4. Assign binary codes to the places in each invariant, in such a way that BDD size is minimized.
5. Given the selected encoding, calculate the transition relation of the PN and the RS by using symbolic traversal techniques.

4 Encoding Safe Petri Nets

This section proposes an encoding scheme that is based on the fact that safe PNs can be decomposed into one-token SM-Components. The places in each SM can be encoded separately using a logarithmic encoding technique. After combining the variables in each invariant, the result is a reduced number of Boolean variables compared to the conventional sparse techniques.

To illustrate the proposed encoding scheme we use the PN in Fig. 2. This PN can be decomposed into six SMs that, in this particular case, cover all places (see Fig. 3). The sparse encoding scheme requires 14 Boolean variables to encode each place, resulting into a density of $\mathcal{D}_{PN} = \lceil \log_2(22) \rceil / 14 = 0.36$.

First, we show how an SM-Component can be encoded by using an optimal number of variables. Then we determine the set of invariants that allows to encode the PN while minimizing the total number of variables. Two methodologies are proposed to select the set of invariants: a simple method that does not consider the interrelations between invariants, and a more elaborated that takes into account those interactions.

4.1 Encoding a Single SM

Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the subset of places covered by a one-token SM-Component I_i. Since one and only one place in $\mathcal{P}_i$ is marked at each marking, a logarithmic encoding can be found for those places. Thus, any injective encoding function $\mathcal{E}_{I_i} : \mathcal{P}_i \rightarrow \mathcal{M}_n$ ($n = \lceil \log_2 |\mathcal{P}_i| \rceil$) is appropriate. Each place must be assigned a unique minterm to uniquely identify the location of the token in I_i, *i.e.*

$$\forall p_j, p_k \in \mathcal{P}_i,\ j \neq k\ :\ \mathcal{E}_{I_i}(p_j) \cdot \mathcal{E}_{I_i}(p_k) = 0\ . \tag{2}$$

4.2 Selecting SMs

The number of variables required to encode a PN directly depends on the selected invariants. Since a place may be covered by different invariants, the *density* of the encoding may decrease if different sets of variables are used to encode the same place at different invariants. To achieve a dense encoding it is important to select a set of invariants that minimize the over-encoding of common places.

Let SMC $= \{I_i\}$ be a set of SMs that (totally or partially) cover the places of the PN. The problem of finding an optimal subset of SMC to encode the PN can be formulated as a *Unate Covering Problem*[14] as follows:

1. Take SMC $\cup\, \mathcal{P}$ as the set of covering objects and $\mathcal{P}$ as the set of covered objects. Each invariant I_i covers a subset of places $\mathcal{P}_i \subseteq \mathcal{P}$. Each place $p_i \in \mathcal{P}$ covers itself.
2. For each $I_i \in$ SMC, define $\mathsf{cost}(I_i) = \lceil log_2(|\mathcal{P}_i|) \rceil$.
3. For each $p_i \in \mathcal{P}$, define $\mathsf{cost}(p_i) = 1$.
4. Find a minimum cost cover of SMs and places.

In practice heuristics are used, e.g. a Fiduccia&Mattheyses algorithm that iteratively takes or rejects invariants for encoding [9]. Obviously, the quality of the final encoding depends on the initial selection of invariants and the order in which they are processed.

The final encoding of each place can be computed as the conjunction of the encoding function used in each particular SM; that is,

$$\mathcal{E}(p_j) = \prod_{I_i : p_j \in \mathcal{P}_i} \mathcal{E}_{I_i}(p_j) \ .$$

The following minimum cost encoding using 10 variables (with density $\mathcal{D}_{PN} = 5/10 = 0.5$) can be found:

- SM_1 covering places $\{p_1, p_2, p_6, p_8\}$ (2 variables).
- SM_3 covering places $\{p_9, p_{10}, p_{12}, p_{14}\}$ (2 variables).
- SM_4 covering places $\{p_9, p_{11}, p_{13}, p_{14}\}$ (2 variables).
- The rest of places encoded with one variable per place (p_3, p_4, p_5 and p_7).

4.3 Combining SMs for a Denser Encoding

The encoding scheme presented in the previous section can be further improved by taking into account that places may be covered by more than one invariant. In that case, a place can be over-encoded, resulting in a less dense encoding scheme. Intuitively, each place only needs to be encoded once even though it can be covered by several SMs.

A denser encoding scheme can be implemented as follows. Let us assume that a subset of SMs, $\{I_1, \ldots, I_{i-1}\}$ is already used to encode some places of the PN. Let us include now an additional SM I_i covering the places $\mathcal{P}_i$. We can partition $\mathcal{P}_i$ into two subsets $\mathcal{P}_i = \mathcal{P}_i^{cov} \cup \mathcal{P}_i^{new}$. $\mathcal{P}_i^{cov}$ contains all those places already covered by $\{I_1, \ldots, I_{i-1}\}$, whereas $\mathcal{P}_i^{new}$ contains the places only covered by I_i.

Given that places in $\mathcal{P}_i^{cov}$ have been already encoded in other SMs, we only need additional variables to encode the places remaining in $\mathcal{P}_i^{new}$; that is, $\lceil \log_2(|\mathcal{P}_i^{new}|) \rceil$ variables. Since most of the SMs of a PN overlap each other, encoding the places in $\mathcal{P}_i^{new}$ rather than the whole set $\mathcal{P}_i$ should lead to much dense encodings.

Once we have determined the number of variables, we need to define the conditions under which binary codes have to be assigned to encode each place. A valid encoding for I_i would be any function $\mathcal{E}_{I_i} : \mathcal{P}_i \rightarrow \mathcal{M}_n$ ($n = \lceil \log_2(|\mathcal{P}_i^{new}|) \rceil$) such that assigns a unique minterm to each place in $\mathcal{P}_i^{new}$; i.e.

$$\forall p_j, p_k \in \mathcal{P}_i^{new}, p_j \neq p_k \ : \ \mathcal{E}_{I_i}(p_j) \cdot \mathcal{E}_{I_i}(p_k) = 0 \ . \tag{3}$$

Equation (3) imposes looser conditions than (2), because no encoding restriction is defined over places already covered in $\mathcal{P}_i^{cov}$. This encoding scheme will use the new Boolean variables to both encode the places in $\mathcal{P}_i^{new}$ and $\mathcal{P}_i^{cov}$. In that way, a certain degree of over-encoding is introduced for places in $\mathcal{P}_i^{cov}$.

Note that for each place $p \in \mathcal{P}_i^{new}$ there may be a set of places $\mathcal{P}_p$ in $\mathcal{P}_i^{cov}$ with the same code as p, i.e.

$$\mathcal{P}_p = \{p' \in \mathcal{P}_i^{cov} \mid \mathcal{E}_{I_i}(p) \cdot \mathcal{E}_{I_i}(p') \neq \emptyset\} \ .$$

This ambiguity is only apparent since the marking of p can be indirectly determined by the marking of the other SMs encoding the places of $\mathcal{P}_p$. In the extreme case of having a single place in $\mathcal{P}_i^{new}$ no additional variables are required because the value of p can be determined by using the places in $\mathcal{P}_i^{cov}$, i.e. p will be marked iff no other place in $\mathcal{P}_i^{cov}$ is marked.

The number of variables required to encode the PN depicted in Fig. 2 can be reduced by using the improved encoding technique. A minimum cost encoding using 6 variables (with density $\mathcal{D}_{PN} = 5/6 = 0.84$) can be found. To derive this encoding all SMs available in Fig. 3 have been used, but only a subset of places in each SM is covered:

- SM_1 covering places $\{p_1, p_2, p_6, p_8\}$ (2 variables).
- SM_2 covering places $\{p_3, p_7\}$ (1 variable).
- SM_3 covering places $\{p_9, p_{10}, p_{12}, p_{14}\}$ (2 variables).
- SM_4 covering places $\{p_{11}, p_{13}\}$ (1 variable).
- SM_5 covering place $\{p_5\}$ (0 variables).
- SM_6 covering place $\{p_4\}$ (0 variables).

Figure 3 shows all SMs of the PN with the suggested codes to be assigned to each place. The encoding described in Table 1 can be derived for the places of the PN maintaining the one-to-one relation between markings and binary codes.

4.4 Characteristic Functions for Places

In general, every place p can be covered by several SM-Components. By using the improved encoding approach presented in the previous section, only one of

Table 1. Encoding for the dining philosophers example in Fig. 2.

SM / place variables	SM1 x_1x_2	SM3 x_3x_4	SM2 x_5	SM4 x_6	SM5	SM6
Encoding	$p_1 = \overline{x_1}\ \overline{x_2}$	$p_9 = \overline{x_3}\ \overline{x_4}$	$p_1 = \overline{x_5}$	$p_9 = \overline{x_6}$	$p_5 = 1$	$p_4 = 1$
	$p_2 = \overline{x_1}\ x_2$	$p_{10} = \overline{x_3}\ x_4$	$p_3 = \overline{x_5}$	$p_{11} = \overline{x_6}$		
	$p_6 = x_1\ x_2$	$p_{12} = x_3\ x_4$	$p_7 = x_5$	$p_{13} = x_6$		
	$p_8 = x_1\ \overline{x_2}$	$p_{14} = x_3\ \overline{x_4}$	$p_8 = x_5$	$p_{14} = x_6$		

Table 2. Characteristic functions for the places according to Table 1.

$\chi[p_1] = \overline{x_1}\ \overline{x_2}\ \overline{x_5}$	$\chi[p_8] = x_1\ \overline{x_2}\ x_5$
$\chi[p_2] = \overline{x_1}\ x_2$	$\chi[p_9] = \overline{x_3}\ \overline{x_4}\ \overline{x_6}$
$\chi[p_3] = \overline{x_5}\ (x_1 + x_2)$	$\chi[p_{10}] = \overline{x_3}\ x_4$
$\chi[p_4] = \overline{x_1}\ \overline{x_3}\ \overline{x_6} + \overline{x_1}\ x_4\ \overline{x_6}$	$\chi[p_{11}] = \overline{x_6}\ (x_3 + x_4)$
$\chi[p_5] = \overline{x_1}\ \overline{x_3}\ \overline{x_5} + x_2\ \overline{x_3}\ \overline{x_5}$	$\chi[p_{12}] = x_3\ x_4$
$\chi[p_6] = x_1\ x_2$	$\chi[p_{13}] = x_6\ (\overline{x_3} + x_4)$
$\chi[p_7] = x_5\ (\overline{x_1} + x_2)$	$\chi[p_{14}] = x_3\ \overline{x_4}\ x_6$

the SMs will be used to encode p, whereas the other SMs will merely assign p a code already used for other places.

Let us call I_p the SM used to encode place p. The characteristic function of place p ($\chi[p]$ markings with p marked) is the following:

$$\chi[p] = \mathcal{E}_{I_p}(p) \cdot \bigwedge_{p' \neq p:\ \mathcal{E}_{I_p}(p) \cdot \mathcal{E}_{I_p}(p') \neq \emptyset} \overline{\mathcal{E}_{I_{p'}}(p')} \tag{4}$$

The characteristic function for each place in Fig. 2 is shown in Table 2.

5 Bounded PN Encoding

This section will show how invariants can be used to efficiently encode any bounded PN. The goal is to characterize the number of tokens in each place by using the information available in a given invariant. Each invariant describes the distribution of tokens in its places for any reachable marking. However, the analysis of token configurations inside a general invariant is more complex that in a simple one-token SM-Component.

To illustrate the proposed encoding scheme we will use the PN depicted in Fig. 1. This PN can be decomposed into the invariants in (1). A sparse scheme requires 10 Boolean variables to encode all places, resulting in a density of $D_{PN} = \lceil \log_2(7) \rceil / 10 = 0.36$.

First, we will analyze which are the reachable markings characterized by each invariant. A number of variables should be assigned to encode each invariant. However, once an invariant has been encoded, less variables are required

to encode the remaining invariants. We introduce a greedy methodology to select which invariants should be encoded first, based on the variable reductions obtained compared to the sparse scheme.

5.1 Token Configurations in Invariants

Let us define a *token configuration* $\mathcal{C}_i$ as an integer assignment to places of an invariant. A token configuration can be *total* or *partial*. A total token configuration defines the token count for all places in the invariant, while a partial token configuration only defines the count for a subset of places. Given the invariant I_1 for the PN in Fig. 1, $\{\{p_1, 1\}, \{p_2, 1\}, \{p_4, 2\}\}$ is a total token configuration, while $\{\{p_1, 1\}, \{p_2, 0\}\}$ is a partial token configuration.

The exhaustive analysis of each invariant provides all possible token configurations that may correspond to potentially reachable markings. The generation of all potential token configurations can be represented as a tree (see Fig. 5), where each node corresponds to a place and the arc to each child is labeled with possible token assignments. Each leaf in the tree represents a total token configuration. In general, we may generate the token configurations of an invariant that has been partially encoded (e.g. see Section 4.3 for safe nets). The subset of the invariant that has been already encoded will be depicted in a rectangular root node in which each outgoing arc to its child is labeled with the number of tokens already assigned to places (in Fig. 5 no place has been encoded, hence 0 is assigned to the root node and its arc). For both invariants I_1 and I_2 in (1) there exists 10 and 9 total token configurations respectively, as shown in Fig. 5.

In order to characterize the token configurations that may lead to potential markings, we define the *potential marking* function for an invariant I_i as:

$$PM_{I_i} \ : \ 2^{\mathcal{P}_i \times \mathbb{N}} \rightarrow \{0, 1\} \ ;$$

where $\mathcal{P}_i$ is the set of places in I_i. The PM function characterizes the token configurations $\mathcal{C}_j \in 2^{\mathcal{P}_i \times \mathbb{N}}$ that satisfy $(PM_{I_i}(\mathcal{C}_j) = 1)$ or not $(PM_{I_i}(\mathcal{C}_j) = 0)$ the invariant, e.g. $PM_{I_1}(\{p_1, 1\}\{p_2, 0\}) = 0$ and $PM_{I_1}(\{p_1, 1\}\{p_2, 1\}\{p_4, 2\}) = 1$ (see Fig. 5). Let $\mathcal{C}_{I_i}$ be the set of potentially reachable total token configurations in I_i.

The combination of information from several invariants further improves the characterization of the potentially reachable markings. Basically, it is known that any reachable marking must agree with all the invariants in the PN. Therefore, if a token configuration does not exists in one invariant then it can not be valid for any other invariant of the PN.

In Fig. 5, $PM_{I_1}(\{p_1, 2\}, \{p_2, 2\}) = 1$ but $PM_{I_2}(\{p_1, 2\}, \{p_2, 2\}) = 0$; therefore the token configuration $\{\{p_1, 2\}, \{p_2, 2\}, \{p_4, 8\}\}$ is not valid for I_1 and we can update the PM function with $PM_{I_1}(\{p_1, 2\}, \{p_2, 2\}, \{p_4, 8\}) = 0$. Similarly, the token configurations between the invariants in Fig. 5 indicates that, in fact, no marking with $\{\{p_1, 2\}, \{p_2, 2\}\}$, $\{\{p_1, 3\}, \{p_2, 1\}\}$ or $\{\{p_1, 1\}, \{p_2, 0\}\}$ could exist. The corresponding arcs in the solution trees are eliminated (denoted by shadowed configurations in Fig. 5). We can conclude that each invariant has 8 possible token configurations.

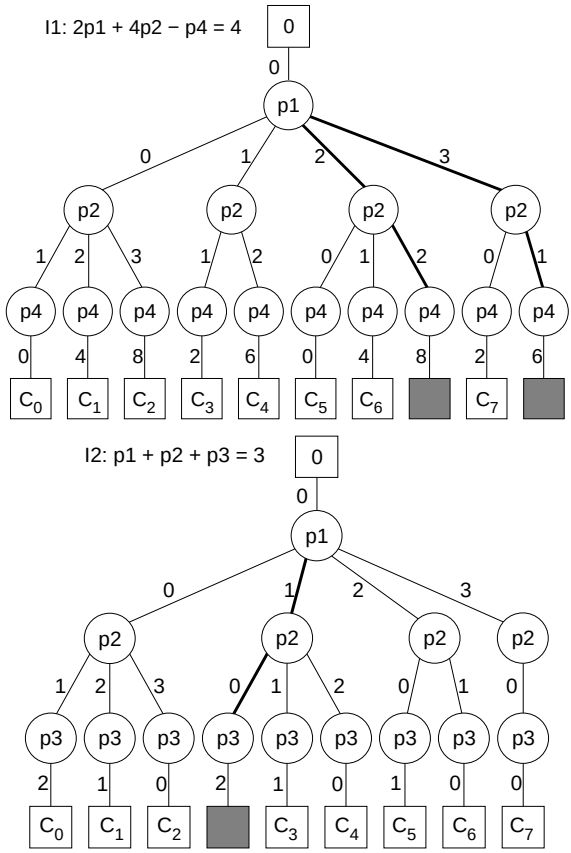


Fig. 5. Token configurations for the invariants of the example in Fig. 1.

Once we have determined the potential token configurations in each invariant we can assign Boolean variables to encode each combination of tokens. The number of variables required to encode the invariant is $V_{I_i} = \lceil \log_2(|\mathcal{C}_{I_i}|) \rceil$. Then, any injective encoding function $\mathcal{E}_{I_i} : \mathcal{C}_{I_i} \rightarrow \mathcal{M}_n$ $(n = V_{I_i})$ is appropriate to encode the invariant. Each different total token configuration must be assigned a unique minterm, *i.e.*

$$\forall \mathcal{C}_j, \mathcal{C}_k \in \mathcal{C}_{I_i},\ j \neq k\ :\ \mathcal{E}_{I_i}(\mathcal{C}_j) \cdot \mathcal{E}_{I_i}(\mathcal{C}_k) = 0\ . \tag{5}$$

For the invariants in (1) we have to encode 8 different token configurations, therefore $\lceil log_2(8) \rceil = 3$ variables are required for each invariant.

Fig. 6 describes a Decision Diagram with one possible encoding of invariant I_1 using three Boolean variables (denoted $x_1 \ldots x_3$). Each one of the 8 total token configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_7)$ is encoded by a different assignment to variables $x_1 \ldots x_3$ (a different minterm described by each branch of the tree). For example,

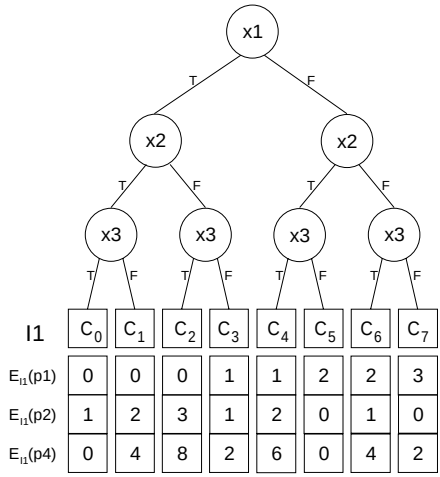


Fig. 6. DDs for the encoding of token configurations in I_1.

the token configuration $\mathcal{C}_4 = \{\{p_1, 1\}, \{p_2, 2\}, \{p_4, 6\}\}$ is encoded as $\mathcal{E}_{I_1}(\mathcal{C}_4) = \overline{x_1}x_2x_3$.

5.2 Invariant Selection for Dense Encoding

Similarly to the techniques used for safe PNs, places that appear in different invariants do not need to be encoded multiple times. Each place must be encoded only at one invariant. The invariant selection process can be formulated as a *Covering Problem* in which each place can be covered by an invariant or left uncovered (using sparse encoding). The goal is to select a number of invariants that minimizes the total number of variables in the encoding.

To avoid the inherent complexity of covering problems, a heuristic algorithm has been derived to select in which invariant a place should be encoded. Basically, we choose the invariant that requires less variables compared to the sparse encoding technique and that has less token configurations to have better control of the minterm assignment process. Given the PN in Fig. 1(a), sparse encoding requires 8 variables for invariant I_1 and 6 variables for I_2. Using the proposed dense encoding only 3 variables are required for each invariant. Invariant I_1 will be encoded first because we obtain an improvement of 5 variables with respect to the sparse technique.

When each potential marking has been encoded it is possible to derive the *encoding function* $\mathcal{E}_{I_i} : \mathcal{P} \times \mathbb{N} \rightarrow \mathcal{F}_n$ that characterizes when a place holds a number k of tokens ($n = V_{I_i}$). This function is the union of total token configurations $\mathcal{C}$ that satisfy $\{p, k\} \in \mathcal{C}$, i.e.

$$\mathcal{E}_{I_i}(p_j, k) = \bigvee_{\mathcal{C}_l \in \mathcal{C}_{I_i} \,:\, \{p_j,k\} \in \mathcal{C}_l} \mathcal{E}_{I_i}(\mathcal{C}_l) \ .$$

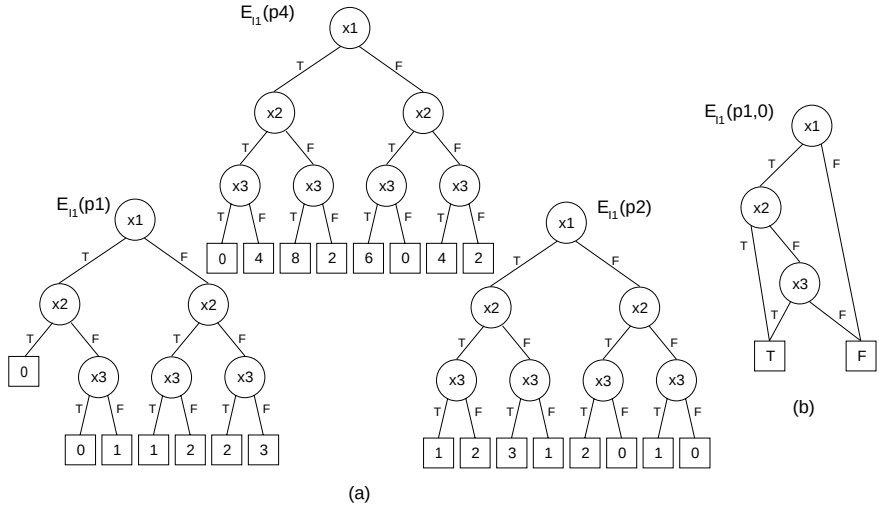


Fig. 7. DDs for the encoding of places in I_1.

A multi-valued encoding function $\mathcal{E}_{I_i} : \mathcal{P} \rightarrow \mathbb{N} \times \mathcal{F}_n$ is defined to characterize all token assignments for each place, i.e.

$$\mathcal{E}_{I_i}(p_j) = \bigvee_{\forall k} [k \times \mathcal{E}_{I_i}(p_j, k)] \ .$$

The token assignments in $\mathcal{E}_{I_i}(p_j)$ can be efficiently represented by means of ADDs. Each branch of an ADD describes a set of binary codes that are assigned to a certain integer value (the token count). Fig. 7(a) explicitly depicts the ADDs for the characteristic function of places in I_1, e.g.

$$\begin{aligned}\mathcal{E}_{I_1}(p_1) &= 0 \times (x_1x_2 + x_1\overline{x_2}x_3) + 1 \times (x_1\overline{x_2}\,\overline{x_3} + \overline{x_1}x_2x_3) + \\ &\quad 2 \times (\overline{x_1}x_2\overline{x_3} + \overline{x_1}\,\overline{x_2}x_3) + 3 \times (\overline{x_1}\,\overline{x_2}\,\overline{x_3}) \ . \\ \mathcal{E}_{I_1}(p_2) &= 0 \times (\overline{x_1}x_2\overline{x_3} + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) + 1 \times (x_1x_2x_3 + x_1\overline{x_2}\,\overline{x_3} + \overline{x_1}\,\overline{x_2}x_3) + \\ &\quad 2 \times (x_1x_2\overline{x_3} + \overline{x_1}x_2x_3) + 3 \times (x_1\overline{x_2}x_3) \ .\end{aligned}$$

On the other side, BDDs are used to represent the subset of markings in which places have a particular token count. Each branch of a BDD describes a set of binary codes that either belongs to the set (if the leaf node is TRUE) or not (the leaf is FALSE). Fig. 7(a) depicts the BDD for the the characteristic function of $\mathcal{E}_{I_1}(p_1, 0) = x_1x_2 + x_1\overline{x_2}x_3$.

Once an invariant has been encoded the rest of invariants may need fewer variables because some places have been already encoded. In the example, places p_1 and p_2 have been already encoded by I_1 and therefore fewer token configurations need to be described when considering I_2. The number of tokens accumulated in $p_1 + p_2$ can be easily computed by operating the ADDs corresponding to $\mathcal{E}_{I_1}(p_1)$ and $\mathcal{E}_{I_1}(p_2)$ [1], i.e.

$$\mathcal{E}_{I_1}(p_1) + \mathcal{E}_{I_1}(p_2) = 1 \times (x_1x_2x_3) + 2 \times (x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3}) +$$

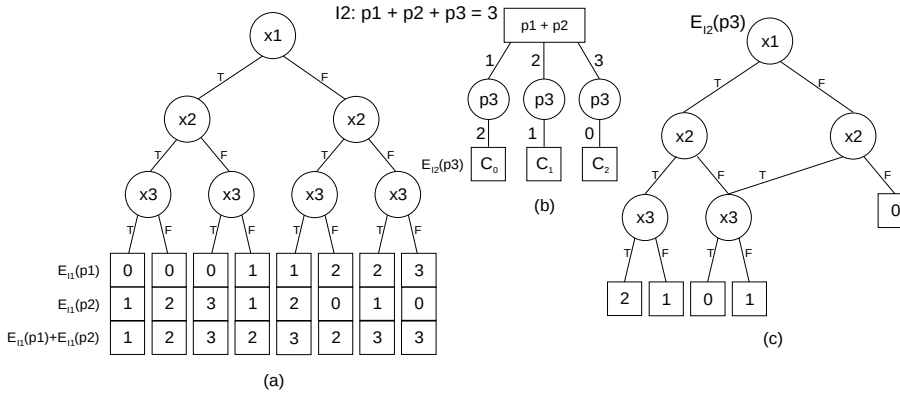


Fig. 8. DDs that characterize the encoding of invariant I_2 after encoding I_1.

$$3 \times (x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) \ .$$

The result shows that only three token configurations exists for the addition of both places, corresponding to $p_1 + p_2 = \{1, 2, 3\}$ (see Fig. 8(a)). Now, it is clear that the value of $\mathcal{E}_{I_2}(p_3)$ can be implicitly derived according to invariant $I_2 : p_1 + p_2 + p_3 = 3$ (see Fig. 8(b)).

The root node (p_1+p_2) of the token configuration tree (see Fig. 8(b)) holds an implicit encoding due to the binary codes previously assigned to other invariants. We denote this encoding function as *implicit encoding function* $\mathcal{E}^i_{I_i} : \mathcal{C}_{I_i} \rightarrow \mathcal{F}_m$ because assigns to each token configuration a function that depends on all the m Boolean variables already assigned in previously considered invariants. In Fig. 8(a) we have that $\mathcal{E}^i_{I_i}(\mathcal{C}_0) = x_1x_2x_3$, $\mathcal{E}^i_{I_i}(\mathcal{C}_1) = x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3}$ and $\mathcal{E}^i_{I_i}(\mathcal{C}_2) = x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$.

Given the root encoding function, the remaining part of the invariants may need fewer variables because the conditions in (5) for the encoding function $\mathcal{E}_{I_i}$ can be relaxed to:

$$\forall \mathcal{C}_j, \mathcal{C}_k \in \mathcal{C}_{I_i},\ j \neq k \ :\ \mathcal{E}^i_{I_i}(\mathcal{C}_j)\mathcal{E}_{I_i}(\mathcal{C}_j) \cdot \mathcal{E}^i_{I_i}(\mathcal{C}_k)\mathcal{E}_{I_i}(\mathcal{C}_k) = 0 \ . \tag{6}$$

Only those token configurations with encoding functions that may intersect should be assigned a unique code (the implicit encoding already prevents some intersections). Hence, the number of variables for encoding is reduced to:

$$V_{I_i} = \left\lceil \log_2 \left| \{ \mathcal{C}_i \ :\ \exists\, \mathcal{C}_j, i \neq j \ s.t.\ \mathcal{E}^i_{I_i}(\mathcal{C}_i) \cdot \mathcal{E}^i_{I_i}(\mathcal{C}_j) \neq 0 \} \right| \right] \ .$$

Finally, if I_p is the invariant used to encode place p, the multi-valued characteristic function $\chi[p]$ of place p must combine the codes assigned in I_p with the implicit information assigned from other invariants, i.e.

$$\chi[p] = \bigvee_{\forall k} \left[k \times \bigvee_{\mathcal{C}_i \in \mathcal{C}_{I_p} \ :\ \{p,k\} \in \mathcal{C}_i} \mathcal{E}^i_{I_p}(\mathcal{C}_i)\mathcal{E}_{I_p}(\mathcal{C}_i) \right] \tag{7}$$

In that case no additional variables are required to encode p_3. The encoding for $\chi(p_3)$ can be created as $\chi(p_3) = 3 - (\mathcal{E}_{I_1}(p_1) + \mathcal{E}_{I_1}(p_2))$ [1], i.e.

$$\chi(p_3) = 0 \times (x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) \\ 1 \times (x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3}) + 2 \times (x_1x_2x_3) \ .$$

The overall encoding process can be described as follows:

1. Compute the potentially token configurations for each invariant.
2. Encode the invariant that provides the maximum variable decrease with respect sparse encoding and minimal number of token configurations.
3. Eliminate invariants with all places already encoded.
4. Update the token configuration trees for the remaining invariants.
5. Repeat from 2 until all places have been encoded.

The encoded reachability graph for the PN in Fig. 1(a) is shown in Fig. 4(b).

6 Computation of Potentially Reachable Markings

Invariants not only provide an efficient mechanism to encode places in a PN, but offer an initial approximation of the *RS*. Any reachable marking must agree with the initial marking at any invariant of a PN. Therefore every invariant can be used to divide the Boolean space into a set of potentially reachable markings and a set of unreachable markings.

The general situation that we consider arises whenever binary codes are left unallocated to any potential token configuration. Given a general invariant I_i : $a_1p_1 + \ldots + a_mp_m = N$, the characteristic function $\chi[I_i]$ of all markings that satisfy that equation is constructed by:

1. Operating the characteristic function $\chi(p_i)$ of all places in the invariant, i.e. $\sum_{p_j \in \mathcal{P}_i} a_i \cdot \chi[p_j]$;
2. In the resulting function, all leaf nodes that are equal to N correspond to markings that satisfy the invariant ($\chi[I_i] = [N - \sum_{p_j \in \mathcal{P}_i} a_i \cdot \chi[p_j] = 0]$).

Since any reachable marking has to satisfy all the invariants, the upper bound of the *RS* is obtained as the conjunction of the characteristic functions for all invariants.

The characteristic function for one token SM-Components can be easily computed by operating the characteristic function of each place. Given an invariant I_i, when a place $p_j \in \mathcal{P}_i$ is marked ($\chi[p_j] = 1$) the rest of places cannot be marked; hence, the characteristic function is computed as:

$$\chi[I_i] = \sum_{p_j \in \mathcal{P}_i} \chi[p_j] \cdot \overline{[\sum_{p_k \in \mathcal{P}_i, k \neq j} \chi[p_k]]} \ .$$

Approximations of the *RS* computed by using structural information improves the symbolic analysis of the PN in two ways:

Table 3. Comparison between sparse and dense encoding schemes for safe PNs.

PN				Sparse encoding				Dense encoding						
name	P	T	RS	V	nTR	nRS	CPU	Ninv	Nnodes	V	nTR	nRTR	nRS	CPU
muller10	40	20	4.2×10^2	40	180	770	1	10	40	20	140	123	189	1
muller20	80	40	2.5×10^5	80	360	3188	9	20	80	40	280	241	668	3
muller30	120	60	6.0×10^7	120	540	6694	51	30	120	60	480	426	1390	13
phil5	65	50	8.5×10^4	65	330	639	2	15	125	25	644	459	158	2
phil10	130	100	7.4×10^9	130	660	7805	40	30	250	50	1284	914	433	24
phil15	195	150	6.4×10^{14}	195	990	87419	700	45	375	75	1924	1369	708	124
slot5	50	50	1.7×10^6	50	330	673	14	10	50	25	325	283	129	5
slot10	100	100	3.8×10^{11}	100	660	2516	1006	20	100	50	650	581	460	309

Table 4. Comparison between sparse and dense encoding schemes for bounded PNs.

PN				Sparse encoding				Dense encoding					
name	P	T	RS	V	nTR	nRS	CPU	Ninv	Nnodes	V	nTR	nRS	CPU
proc1	8	8	7	11	107	29	0	4	41	5	131	12	0
robot1	17	8	1.6×10^2	28	208	389	1	11	222	12	99	58	1
robot2	15	6	4.8×10^1	24	149	243	1	10	69	6	817	9	1
robot12	24	14	1.3×10^3	40	358	1330	2	13	9465	18	647	141	7

- A set of markings that is known to be unreachable offers a number of binary codes to be used as *don't care* set. The BDD representation of functions involved in the symbolic analysis can be simplified by using this *don't care* set. In particular, the size of the transition relation and the *RS* of the PN can be reduced.
- The potential *RS* approximations may already provide enough information to determine if the properties under analysis are satisfied in a positive or negative way without requiring the symbolic traversal of the PN.

7 Experimental Results

The efficiency of the proposed encoding technique is measured in terms of reductions achieved for number of variables, BDD nodes to represent the transition relation and the *RS* of the PN, and CPU computation times.

Table 3 compares the results of symbolic traverse after both sparse and dense encoding of several safe PNs based on the general invariant-based algorithm. Scalable examples have been used. *Muller* describes a Muller pipeline with n-stages, *Phil* describes n competing philosophers, *Slot* a model for the slotted-ring protocol with n stages. We have analyzed the results obtained by using a sparse encoding (labeled Sparse) and a dense encoding with set of minimal invariants computed with algebraic techniques (labeled Dense). For both cases we provide the number of Boolean variables required by the encoding (V), the number of BDD nodes to represent the transition relations (nTR) and the RS (nRS), and the computation times (CPU). Additionally, for the dense encoding we provide the number of invariants that have been used ($Ninv$) and the total number of token assignments generated along the encoding process ($Nnodes$). When using

the potentially reached markings to simplify the TR of the PN the number of BDD nodes is also presented ($nRTR$).

The experiments show that 50% variable reductions or better can be obtained. The influence of these results is evident on the number of BDD nodes to represent the RS (70% to 90% are obtained) and on the computation times (40% to 80% speedups are achieved). Conversely, the number of BDD nodes to represent the transition relations may even increase due to the complexity of the encoding assignments. The computation of the potentially reachable markings also help to further reduce the size of the TRs between $10 - -30\%$.

Table 4 compares the results of symbolic traverse after both sparse and dense encoding of a few bounded PNs. The examples describe several robot control automatons. Again 50% variable reductions can be obtained. The influence of these results is also quite significant on the number of BDD nodes to represent the RS. However, the increase in the number of nodes to represent the transition relations may reduce the computation speed-ups. Further research is needed in that direction. From the *robot*12 example it can also be seen that in same cases the number of token configurations may be even bigger that the reachable states. Heuristics must be derived to avoid exploring invariants with large number of configurations.

8 Conclusions

This paper has presented encoding techniques that improve the efficiency of symbolic methods for the analysis of PNs. Structural PN theory provides sets of P-invariants to identify interrelations among places, which allows to immediately identify sets of unreachable markings. These techniques alleviate the complexity of the existing symbolic techniques for the calculation of the exact reachability set.

The structural theory of PNs goes beyond P-invariants. Although the structure is not enough for the exact analysis of a PN, it provides information that can be efficiently combined with symbolic analysis. Future work intends to derive a general framework to combine the efficiency of the structural PN theory with the accuracy of the symbolic techniques.

Acknowledgments

This work has been partially funded by CICYT TIC 98-0410 and ACiD-WG (ESPRIT 21949). We thank anonymous reviewers for their useful comments.

References

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proc. ICCAD*, pages 188–191, November 1993.

[2] K. S. Brace, R. E. Bryant, and R. L. Rudell. Efficient implementation of a BDD package. In *Proc. DAC*, pages 40–45, 1990.

[3] F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publishers, 1990.

[4] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] Jerry R. Burch, Edmund M. Clarke, D. E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Trans. on CAD*, 13(4):401–424, 1994.

[6] G. Chiola Compiling Techniques for the Analysis of Stochastic Petri Nets. In *4th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 11–24, September 1989.

[7] J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, Cambridge, Great Britain, 1995.

[8] J. Desel, K.P. Neuendorf, and M.D. Radola. Proving nonreachability by modulo-invariants. *Theoretical Computer Science*, (153):49–64, 1996.

[9] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. DAC*, 1982.

[10] M. Hack. *Analysis of production schemata by Petri nets*. M.s. thesis, MIT, February 1972.

[11] G. J. Holzmann. An Improved Protocol Reachability Analysis Technique *Software Practice and Experience*, 18(2):137–161, 1988.

[12] R. Kannan and A. Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM J. Comput.*, 4(8):499–577, 1979.

[13] K. Lautenbach. Linear algebraic techniques for place/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986*, volume 254 of *LNCS*, pages 142–167. Springer-Verlag, 1987.

[14] E. J. McCluskey. Minimization of boolean functions. *Bell Syst. Technical Journal*, (35):1417–1444, November 1956.

[15] G. Memmi and G. Roucairol. Linear algebra in net theory. In W. Brauer, editor, *Net Theory and Applications*, volume 84 of *LNCS*, pages 213–223. Springer-Verlag, 1980.

[16] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.

[17] E. Pastor and J. Cortadella. Efficient encoding schemes for symbolic analysis of petri nets. In *Proc. Design, Automation and Test in Europe*, pages 790–795, Paris (France), February 1998.

[18] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri net analysis using boolean manipulation. In *15th Int. Conf. on Application and Theory of Petri Nets*, volume 815 of *LNCS*, pages 416–435. Springer-Verlag, June 1994.

[19] T. Yoneda, H. Hatori, A. Takahara, and S. Minato. BDDs vs. Zero-Suppressed BDDs: for CTL symbolic model checking of petri nets. In *Proc. of Formal Methods in Computer-Aided Design*, volume 1166 of *LNCS*, pages 435–449. Springer-Verlag, 1996.

Stubborn Sets for Standard Properties

Karsten Schmidt

Institut für Informatik, Humboldt–Universität zu Berlin, 10099 Berlin, Germany

Abstract. Stubborn sets are a tool for state space reduction preserving certain system properties. We present stubborn set approaches for all popular Petri net standard properties. This extends the list of properties that can be analysed successfully (including boundedness, reversibility). For other properties, our approach can lead to larger reductions (reachability) than previous ones. Furthermore, shortest and cheapest witness paths for several properties are now preserved.

Keywords: (Theory) Analysis of nets, Computer tools for nets

1 Introduction

Using stubborn sets, situations where sequences of transitions can fire in arbitrary order are detected. By firing as few as possible of the possible permutations, the size of state spaces can decrease significantly. Reduction is obtained by firing, at a marking m, only some transitions at each state, collected in the stubborn set $St(m)$. The *art of stubborn sets* is to keep the reduced size as small as possible, but large enough to preserve a given property of the underlying system.

The first, and still most popular, application field of stubborn sets was the dead marking problem [9]. Then, eliminating the problem of ignored transitions, other standard properties like liveness and dead transitions could be preserved [10]. The stubborn set concept was generalised to language preservation [10] and linear time temporal logic (LTL) model checking [11, 12, 6, 3]. These two approaches depend on the distinction between visible and invisible transitions, i.e. transitions that do or do not influence the desired property. The language of visible transitions is fully preserved while state space reduction concerns only invisible transitions. Though visibility can be relaxed in some situations [5, 17], the general approaches do not work very well for global or almost global properties (properties with a small number of invisible transitions). Furthermore, care must be taken to the ignoring problem.

Attempts to apply stubborn sets to model checking for the branching time logic CTL [2] lead to significantly stronger restrictions (i.e. larger stubborn sets) than LTL–preserving methods.

In [13–15], many derivatives of the stubborn set method are surveyed.

We propose another policy of stubborn set creation. Where most existing methods consider stubborn sets as a superset of a *single* enabled transition (arbitrarily chosen), our stubborn sets are supersets of *attractor sets* of transitions.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 46-65, 1999.
© Springer-Verlag Berlin Heidelberg 1999

These attractor sets play a key role for directing the generation of the reduced graph into the desired direction. This concept is also present in the *tester* concept in [12] where the actions of the tester process are closely related to our attractor sets. However, the tester concept treats visible transitions in a similar way as language preserving stubborn sets and is therefore not suitable for global properties.

We present our method property by property. Thereby we start with boundedness and reachability. Having done this, we generalise the pattern of our methodology. Then we discuss invariance and satisfiability of state predicates on a more general level. We continue with the preservation of shortest and cheapest paths. After that we revisit the dead transition and dead marking problems. For preserving liveness and reversibility, we study stubborn sets in the context of strongly connected components of the reachability graph. We close our list of properties with the (more complicated) home state problem.

2 Petri nets

For the purpose of simplicity, we present the approach for place/transition nets. However, the idea can be easily transferred to other formalisms that have a concept of stubborn sets.

Definition 1 (Petri net). *A tuple* $N = [P, T, F, W, m_0]$ *is a Petri net iff* P *and* T *are finite, nonempty, and disjoint sets (of* places *and* transitions*),* $F \subseteq (P \times T) \cup (T \times P)$ *(the set of* arcs*),* $W : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ *such that* $W([x,y]) > 0$ *iff* $[x,y] \in F$ *(the arc* multiplicities*), and* m_0 *is a* marking*, i.e. a mapping* $m_0 : P \longrightarrow \mathbb{N}$.

For a place or transition x, ${}^\bullet x = \{y \mid [y,x] \in F\}$ denotes the *pre–set* of x, and $x^\bullet = \{y \mid [x,y] \in F\}$ denotes its *post–set*.

Definition 2 (Transition relation). *We say:* t *can fire at a marking* m *yielding a marking* m' *(written:* $m \xrightarrow[R]{t} m'$*) iff for all* $p \in P$, $m(p) \geq W([p,t])$ *and* $m'(p) = m(p) - W([p,t]) + W([t,p])$.

If there exists a m' to a given m and t such that $m \xrightarrow[R]{t} m'$, then we say: t is enabled at m. We extend the transition relation to sequences of transitions. Define $m \xrightarrow[R]{\varepsilon} m$ for arbitrary m and the empty sequence ε, and $m \xrightarrow[R]{wt} m'$ (w being a transition sequence and t a transition) iff there is a m^* such that $m \xrightarrow[R]{w} m^*$ and $m^* \xrightarrow[R]{t} m'$. If there is a transition sequence w such that $m \xrightarrow[R]{w} m'$, we write $m \xrightarrow[R]{*} m'$.

Definition 3 (Reachability graph). *A directed labelled graph is the reachability graph of a Petri net* $N = [P, T, F, W, m_0]$ *iff its set of nodes is the set of all reachable markings, i.e.* $\{m \mid m_0 \xrightarrow[R]{*} m\}$*, and* $[m, m']$ *is an edge labelled with* t *iff* $m \xrightarrow[R]{t} m'$.

We generalise the concept of reachability graphs such that reduced reachability graphs are covered.

Definition 4 (Transition system). *Let M be a set of markings of a Petri net N containing its initial marking and $\cdot \dot{\rightarrow} \cdot$ a relation on $M \times T \times M$ such that $m \xrightarrow{t} m'$ implies $m \xrightarrow[R]{t} m'$. Then $[M, \cdot \dot{\rightarrow} \cdot]$ is a transition system of N.*

For any finite set X, let X^* be the set of finite words over X. We extend the general transition relation $\cdot \dot{\rightarrow} \cdot$ to the domain $M \times T^* \times M$ in the same manner as the original reachability relation.

3 Stubborn sets

Stubborn sets of transitions enjoy a certain independency from the remaining transitions. There are two different degrees of independency leading to different definitions.

Definition 5 (Stubborn set). *Let m be a marking. Then a set $St(m)$ of transitions is* stubborn in the weak sense *at m iff for every sequence w of transitions in $T \setminus St(m)$ and every $t \in St(m)$ it holds: If there is a m' such that $m \xrightarrow[R]{wt} m'$ then $m \xrightarrow[R]{tw} m'$. $St(m)$ is* stubborn in the strong sense *iff additionally for every sequence w of transitions in $T \setminus St(m)$ and every $t \in St(m)$ it holds: If there are m_1 and m_2 such that $m \xrightarrow[R]{w} m_1$ and $m \xrightarrow[R]{t} m_2$ then there is an m' such that $m \xrightarrow[R]{wt} m'$.*

While weak stubborn sets require the ability to sort stubborn transitions to the beginning of a sequence, strong stubborn sets require the exchangeability of stubborn and non–stubborn transitions in both directions.

In many papers, sets as defined above are called *semistubborn*. For our approach, a distinction between semistubborn and stubborn sets is not necessary, so we use the shorter name. Note that in our setting the empty set of transitions as well as the set of all transitions are always stubborn in both senses at any marking.

Given a mapping St assigning a stubborn set to every marking m, we can construct a reduced reachability graph in the same way as the full reachability graph. All we have to do is to replace the original reachability relation $m \xrightarrow[R]{t} m'$ by a new relation $m \xrightarrow[St]{t} m'$, where $m \xrightarrow[St]{t} m'$ iff $m \xrightarrow[R]{t} m'$ and $t \in St(m)$. Building a graph starting from m_0 and using this relation, we obtain a transition system being a subgraph of the reachability graph.

If we assign the set of all transitions as stubborn set to all markings, then the reduced graph is identical to the full graph. If we assign the empty set to m_0, then the reduced graph contains only m_0 itself. It is difficult to deduce interesting

properties from the latter graph. Thus, the art of applying stubborn sets is to choose stubborn sets that are large enough to preserve certain properties, but as small as possible in order to obtain a significant reduction of the search space.

For this purpose we come up with a given set $A(m)$ of transitions that may or may not depend on m and use a (preferably small) stubborn superset of $A(m)$. We denote a stubborn superset of $A(m)$ at a marking m by $St(m, A)$. The set $A(m)$ shall be used for tuning the stubborn set such that it preserves the desired property. We call $A(m)$ an *attractor set*. We refer to a reduced graph obtained by using $St(m, A)$ at every marking by using the relation symbol $\cdot \xrightarrow[A]{} \cdot$.

Stubborn supersets can be calculated as a closure operation. Starting with $A(m)$, we include step by step such transitions that could violate the required property for stubborn sets. The correctness of our results does not depend on the particular stubborn set algorithm used. Thus, we refer the reader to [9, 17] for calculation issues. According to [17], stubborn sets can also be calculated by removing transitions step by step from the set of all transitions. This algorithm can also calculate stubborn supersets of A since it can mark some transitions as unremovable.

4 The ignoring problem

For some properties (for example, liveness, reversibility) it shall be necessary to consider the ignoring problem. These properties as well as ignoring are related to the strongly connected components of the reachability graph. We study the ignoring problem only for bounded nets, i.e. finite state systems.

Definition 6 (Strongly connected components). *Let $[M, \cdot \rightarrow \cdot]$ be a transition system. Two markings m and m' are mutually reachable ($m \leftrightharpoons m'$) iff $m \xrightarrow{*} m'$ and $m' \xrightarrow{*} m$. A strongly connected component (SCC) of $[M, \cdot \rightarrow \cdot]$ is an equivalence class of M with respect to $\leftrightharpoons$. A terminal strongly connected component (TSCC) is a strongly connected component C where $m \in C$ and $m \xrightarrow{*} m'$ imply $m' \in C$.*

Finite transition systems have at least one TSCC. TSCC are the maximal elements of the reachability relation between SCC (define $C_1 \rightarrow C_2$ iff there exist $m_1 \in C_1$ and $m_2 \in C_2$ such that $m_1 \rightarrow m_2$). Reachability between SCC is a reflexive partial order relation.

Definition 7 (Ignored transition). *Let $[M, \cdot \rightarrow \cdot]$ be a transition system, C one of its TSCC, and t a transition. If for all $m \in C$, t is enabled at m but for no m' the arc $m \xrightarrow{t} m'$ is contained in the transition system, then t is ignored.*

Using TARJAN's algorithm for the detection of SCC during depth first graph exploration, it is easy to construct a graph without ignored transitions. Immediately after the last marking m of a SCC is entered the last time, the SCC is recognised and can be identified as terminal or non–terminal. If it is terminal, we can check for ignored transitions. If there are any, we can enlarge the set of

transitions to be fired at m without threatening the depth first nature of graph exploration.

Avoiding ignored transitions aims in the following property, proved in [10]:

Proposition 1 (Ignored transitions and TSCC). *If a transition system is reduced with respect to stubborn sets in the strong sense and does not contain ignored transitions, then for every TSCC C of the full reachability graph there is a $m \in C$ contained in the reduced transition system.*

5 Existing stubborn set approaches

We give a brief overview on some relevant stubborn set methods in the context of standard properties.

Basic stubborn sets use weak stubborn sets containing an enabled transition without further restrictions. They preserve all dead markings and one infinite path if there is one in the full graph. No other properties are preserved though some (for instance, reachability) can be traced back to the dead marking problem by some net transformations.

Extended stubborn sets use strong stubborn sets and remove ignored transitions. This way, in addition to basic stubborn sets, we can preserve dead transitions and liveness.

Language preserving stubborn sets assign a letter or the empty word to every transition. The transitions that are labelled by the empty word are called *invisible*, the remaining ones are visible. The language of the net is the set of all sequences of labels of *finite* firing transition sequences. Using extended stubborn sets with the further restriction that a stubborn set either does not contain enabled visible transitions or includes all visible transitions, the language of the net in the above sense is preserved. This technique can be used for the preservation of many safety properties.

LTL–preserving stubborn sets preserve formulas of a next–step free linear temporal logic. They are based on language preserving stubborn sets where all transitions that can potentially change the value of atomic propositions in the given formula are treated as visible. Additionally it is required that for every visible transition t and every infinite path of the reduced graph (in particular, every elementary cycle) there is at least one state where t is in the stubborn set used.

CTL–preserving stubborn sets preserve formulas of the branching time temporal logic CTL. They are based on language preserving stubborn sets and obey the additional restriction that a stubborn set either has only one enabled transition which is invisible or contains all enabled transitions. This restriction guarantees that the branching structure of the full graph is reflected correctly in the reduced graph.

The methods are characterised by an increasing number of requirements. All requirements potentially lead to less reduction.

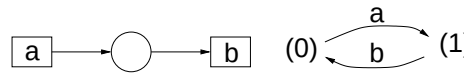


Fig. 1. Unbounded net with finite reduced graph

6 Boundedness

A net is bounded iff its number of reachable markings is finite. This is the case iff there is an upper bound for the number of tokens in the net. Preserving boundedness means that the reduced graph is finite iff the full graph is.

Definition 8 (Boundedness). *Let $N = [P, T, F, W, m_0]$ be a Petri net and $[M, \cdot \stackrel{\cdot}{\rightarrow} \cdot]$ a transition system of N. A place $p \in P$ is bounded iff there is a number c such that for all m, $m_0 \stackrel{*}{\rightarrow} m$ implies $m(p) < c$. A net is bounded iff there is a number c such that for all m, $m_0 \stackrel{*}{\rightarrow} m$ implies $\sum_{p \in P} m(p) < c$.*

Basic and extended stubborn sets do not preserve boundedness. Consider Figure 1 (from [18]).

At the initial marking, $\{a\}$ is stubborn. At the other marking, $\{b\}$ is stubborn. Thus, the reduced graph is finite though the net is obviously unbounded. No transition is ignored.

LTL and CTL preserving approaches are not applicable to the boundedness problem since boundedness cannot be expressed as a formula of LTL or CTL. Language preserving stubborn sets can be used in principle but require extended stubborn sets and must treat all transitions that change the number of tokens in the net as visible.

We aim at constructing a reduced reachability graph that is finite if and only if the original graph is. Thus, at any marking, we want to keep the reachable *larger* markings inside the reduced graph, while we are not interested in the remaining markings. Between any "current" marking and a larger marking, we always find a transition where more tokens are produced than consumed. Transitions of this kind form the key to the reduced graph.

Let $N = [P, T, F, W, m_0]$ be a Petri net and A be the set of all transitions t in T where $\sum_{[p,t] \in F} W([p,t]) < \sum_{[t,p] \in F} W([t,p])$. A does not depend on a particular marking. Nevertheless we use it as attractor set. Consider the reduced graph obtained by using a weakly stubborn superset of A at every marking. Then it holds:

Theorem 1 (Preserve boundedness). *The full reachability graph is finite if and only if the reduced graph as described above is finite.*

Proof. If the original graph is finite, then the reduced graph as a subgraph is finite as well.

Let the original graph be infinite and assume that the reduced graph is finite. This means that there is a number c such that for all markings m appearing in

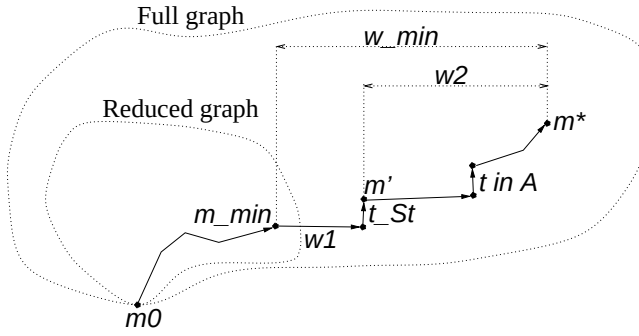


Fig. 2. Situation studied in the proof of theorem 1

the reduced graph (i.e. satisfying $m_0 \xrightarrow[A]{*} m$), $\sum_{p\in P} m(p) \leq c$ while in the original graph, there is a marking m^* such that $m_0 \xrightarrow[R]{*} m^*$ and $\sum_{p\in P} m^*(p) > c$. m_0 appears as well in the reduced graph. Thus, there are markings m and transition sequences w such that $m_0 \xrightarrow[A]{*} m$ and $m \xrightarrow[R]{w} m^*$. Select among all those m and w such m_{min} and w_{min} that length of w_{min} is as small as possible. Since $\sum_{p\in P} m_{min}(p) \leq c$ and $\sum_{p\in P} m^*(p) > c$, an element of A must appear in w_{min}. Therefore w_{min} contains occurrences of transitions in $St(m, A)$. Thus, we can find a marking m', a transition $t_{St} \in St(m, A)$ and transition sequences w_1 and w_2 such that $w_{min} = w_1 t_{St} w_2$, $m \xrightarrow[R]{w_1 t_{St}} m' \xrightarrow[R]{w_2} m^*$ and w_1 does not contain elements of $St(m, A)$ (i.e. t_{St} is the first occurrence of a transition in $St(m, A)$ in w_{min}). Note that t_{St} is not necessarily an element of A itself. The situation we have found so far is illustrated in figure 2. According to Definition 5, there is an m_1 such that $m \xrightarrow[R]{t_{St}} m_1 \xrightarrow[R]{w_1} m'$. Since t_{St} is in $St(m, A)$, m_1 is contained in the reduced graph. Furthermore, there is a path from m_1 to m^*, namely $w_1 w_2$. See figure 3 for an illustration of the last conclusions. The path $w_1 w_2$ is shorter than w_{min}, in contradiction to the selection of w_{min}. The only assumption that could be responsible for this contradiction is the boundedness of the reduced graph. □

In the example at the beginning of the section, we would always choose $\{a\}$ as stubborn set, since it is the only transition that increases the number of tokens and is stubborn.

The choice of relevant transitions can be refined when structural knowledge is available. For instance, if we know a set of semi–positive place invariants, we can replace the set A by the set of all pre–transitions of places not covered by an invariant. The proof follows the same pattern: prove that between the current marking and a larger marking there is a transition of the new attractor set, and therefore there is an element of the stubborn set, too, included in the path. Consequently, there is always a shorter path than the selected minimal one.

The largest reduction can be obtained in those cases where A is empty. In this case, we can use the empty set as stubborn set which means that the reduced

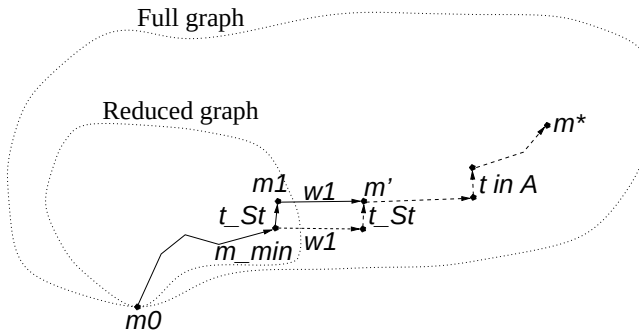


Fig. 3. Main conclusion in the proof of theorem 1

graph consists of m_0 only. This is nevertheless correct since in this case the net is conservative and consequently bounded. The controlled use of empty stubborn sets or sets without enabled transitions is an important difference between the original and our stubborn set approach.

Our reduced graph preserves boundedness of a net, but not necessarily boundedness of particular places. For instance, if we add an output place to transition b of our introductory example, our reduced graph would always fire a and leave the new place unmarked. Therefore, preserving boundedness of particular places requires to calculate a reduced graph with respect to another set of relevant transitions.

Theorem 2 (Preserve boundedness of single places). *Let p be a place of a net $N = [P, T, F, W, m_0]$ and m a marking. If a reduced graph is calculated by firing a weakly stubborn superset $St(m, {}^\bullet p)$ of ${}^\bullet p$ at m, then p is bounded in the reduced graph if and only if it is bounded in the original graph.*

The proof follows the same pattern as the proof concerning boundedness of the net. Instead of showing that an element of A appears on a path between the current and any target marking, we show that an element of ${}^\bullet p$ appears there. Call ${}^\bullet p$ an attractor set for the single place boundedness problem.

In combination with the coverability graph method [4, 1], the reduced graph generation can be used to decide boundedness of places of the net. The combined approach works exactly like usual coverability graph generation [4, 1] but considers only a stubborn set of transitions at every generalised marking (generalised markings do not impose a problem to stubborn set generation since enabledness is clearly defined).

For preserving boundedness of particular places, [16] suggests to consider the formula $\exists k(\Box m(p) \leq k)$. Thereby $\Box$ is the *always* operator of linear time temporal logic. The existential quantifier is not part of the logic, therefore the formula must be treated as an infinite disjunction. Fortunately, all parts of the disjunction share the same set of visible transitions, namely ${}^\bullet p \cup p^\bullet$. This opens the way to use language or LTL

preserving stubborn sets for the bounded place problem. However, this proposal requires further investigation since we deal with potentially unbounded systems where the combination with coverability graphs is compulsory. It is not obvious how the ignoring problem is solved since the coverability graph does not completely reflect the SCC structure of the original graph. For our approach, weak stubborn sets are used in the basic sense, i.e. no information about the graph structure is involved.

7 Reachability

Let m^* be a marking, called the target marking. We aim at constructing a reduced graph that contains m^* if and only if the full graph does.

Language, LTL and CTL preserving approaches do not work well on this problem since there are no invisible transitions with respect to the reachability of m^* (except in the boring case of transitions that do not change a marking at all).

The reachability problem can be transformed into a dead marking problem. This method is currently implemented in INA [7]. An additional transition t^* kills the net as soon as the target marking is covered. For this purpose, t^* has an arc from every place p with multiplicity $m^*(p)$. An additional place p^* is necessary having arcs in both directions to the transitions without input places and an output arc to t^*. m^* is reachable in the original net iff the empty marking appears in the set of dead markings of the transformed net. However, there are several examples (one is presented later in this section) where this transformation does not yield any reduction.

Finally, for nets with known capacities for all places the fact technique can be used to transform a reachability problem into a dead transition problem. Then, extended stubborn sets can be used while basic stubborn sets are not sufficient.

Let m be a marking of the reduced graph. We have to calculate a suitable stubborn set at m. If $m = m^*$, then nothing is left to do. We can use $\emptyset$ as stubborn set. Otherwise, there is a place p where $m(p) \neq m^*(p)$. Select an arbitrary place p with this property. If $m(p) > m^*(p)$, let $A(m) = p^\bullet$. If $m(p) < m^*(p)$, let $A(m) = {}^\bullet p$. It is sufficient to use stubborn supersets of $A(m)$ in the weak sense.

Theorem 3 (How to preserve reachability). *If a reduced graph is constructed using the stubborn sets described above, then the reduced graph contains m^* if and only if the original graph does.*

Proof. If m^* is not reachable in the original transition system, then it cannot be reachable in the reduced system, since the reduced transition system is a subsystem of the full one.

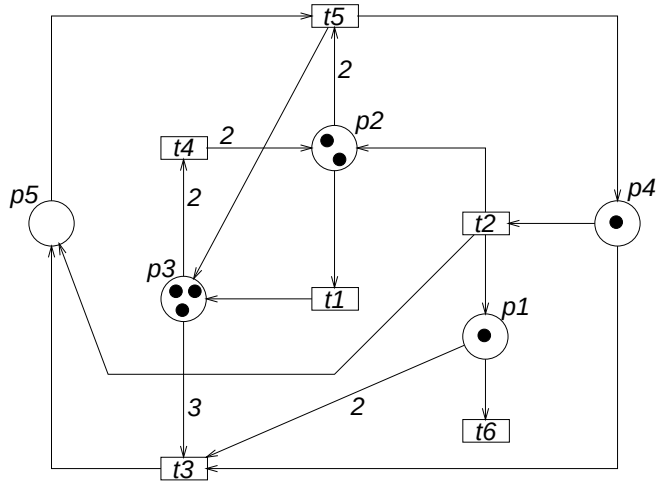


Fig. 4. A net coding a non–trivial reachability problem

Assume, m^* is reachable in the full graph, but not in the reduced graph. Since at least m_0 is contained in the reduced graph and $m_0 \xrightarrow[R]{*} m^*$, we can select such marking m and transition sequence w that $m_0 \xrightarrow[A]{*} m$, $m \xrightarrow[R]{w} m^*$, and w has the shortest possible length. Consider the stubborn set $A(m)$ used at m. If $m = m^*$, nothing remains to show. Otherwise we selected a place p where $m(p) < m^*(p)$ or $m(p) > m^*(p)$. In the first case, sequence w must contain a transition producing tokens on p. In the second case it must contain a transition removing tokens from p. Therefore, in both cases w contains transitions from $A(m)$. Thus, we may divide w into $w_1 t w_2$ where t is the first appearance of a stubborn transition at m and w_1, w_2 are transition sequences. The definition of stubborn sets states the existence of a marking m_1 such that $m \xrightarrow[A]{t} m_1$ and $m_1 \xrightarrow[R]{w_1 w_2} m^*$. $w_1 w_2$ is obviously shorter than w, in contradiction to the selection of w. □

Example. Figure 4 depicts an unbounded net. According to [8], this net was introduced by H. MÜLLER in the early 80ies. Reachability of some markings in this net is not easy to see, since (as the following considerations shall show) it is necessary to go "to the wrong direction" first before reaching a certain marking, which appears to be a difficult situation for our approach.

We want to analyse the reachability of the marking $(2, 2, 0, 0, 1)$ from the depicted marking $(1, 2, 3, 1, 0)$. This marking is indeed reachable, for instance via the following shortest path:

$(1,2,3,1,0) \xrightarrow[R]{t2} (2,3,3,0,1) \xrightarrow[R]{t1} (2,2,4,0,1) \xrightarrow[R]{t4} (2,4,2,0,1) \xrightarrow[R]{t4} (2,6,0,0,1)$
$\xrightarrow[R]{t5} (2,4,1,1,0) \xrightarrow[R]{t2} (3,5,1,0,1) \xrightarrow[R]{t5} (3,3,2,1,0) \xrightarrow[R]{t2} (4,4,2,0,1) \xrightarrow[R]{t5} (4,2,3,1,0)$
$\xrightarrow[R]{t3} (2,2,0,0,1)$.

Though the number of tokens on $p2$ is at m_0 already the desired one for the target marking, it is necessary to put up to 6 tokens on it before reaching the target marking. The marking cannot be reached without going this roundabout.

Using breadth–first search in the full graph, we have to develop the graph up to depth 10 to find the marking. At that stage, 91 nodes have been computed. INA tries to modify the net such that the original reachability problem is translated into a reachability problem concerning a dead marking and performs breadth–first search on the reduced graph of the modified net. INA calculates 94 nodes for the problem (this means, no reduction is obtained while a few additional dead markings are generated that correspond to markings strictly covering the target marking). With our approach, we had to search in a reduced graph where only 51 nodes were computed with a depth of 10 or less (the number varies a bit according to the strategy to select the difference place p for stubborn set calculation, but the reduction remains roughly the same).

8 The pattern of attractor sets

In all approaches up to now (and in the sequel), we used stubborn supersets of a certain set A that does (reachability) or does not (boundedness) depend on the current marking. In all cases, the used set consists of exactly those transitions that would, when fired, bring us closer to the desired situation (add tokens to the net or a particular place, change the number of tokens on a place such that it becomes closer to the desired number). In this sense, attractor sets guide graph generation into the desired direction. It is the task of the stubborn closure to take care about those situations where the desired transitions cannot fire immediately but only after having fired some other transitions.

9 Satisfiability and Invariance

We generalise the stubborn set approach for reachability to the satisfiability of state predicates.

Definition 9 (State predicate). *Let N be a Petri net. Any boolean combination of strings of the form pRk where p is a place of N, $R \in \{<, >, \geq, \leq, =, \neq\}$, and k is a natural number, is called a state predicate for N. A state predicate is satisfiable iff there is a marking m reachable from the initial marking such that replacing the place names p in the predicate by the corresponding values $m(p)$ transforms the predicate into a true statement (using the standard interpretation for the relation symbols). A state predicate is an invariant iff all replacements according to reachable markings transform it into a true statement.*

Given a target marking m^*, reachability of m^* is equivalent to the satisfiability of $\bigwedge_{p \in P} p = m^*(p)$.

We assume that conjunction ($\wedge$) and disjunction ($\vee$) are the only boolean operators used. This does not restrict generality, since negation symbols atop of boolean operators can be removed using DE MORGAN's rules while negation symbols atop of atomic comparisons can be removed, since our set of allowed comparisons is closed under negation. Other than this restriction, we do not assume any normal form. Thus, we do not suffer from the well known complexity problems of normal form construction.

In the sequel, we study only satisfiability of a given state predicate. However, satisfiability and invariance can be translated into each other by negating the predicate.

As for reachability, we try to identify a (small) set of transitions, such that one of these transitions necessarily occurs on a path between the current marking and any satisfying marking.

Let m be an arbitrary marking. If m satisfies the predicate, then nothing is left to do. We use $\emptyset$ as stubborn set. If m does not satisfy the predicate, then we define the attractor set $A_\phi(m)$ inductively. For every m that does not satisfy ϕ, we determine $A_\phi(m)$ such that every path from m to a marking satisfying ϕ contains a transition in $A_\phi(m)$. When $A_\phi(m)$ enjoys this property, and stubborn supersets of $A_\phi(m)$ (in the weak sense) are used for state space reduction then ϕ is satisfiable in the reduced graph iff it is satisfiable in the original graph. The proof is basically the same as for reachability.

If ϕ is an atomic comparison, $A_\phi(m)$ can be determined as follows:

Formula ϕ	Attractor set $A_\phi(m)$
$p < k$	$p^\bullet$;
$p > k$	${}^\bullet p$;
$p \leq k$	$p^\bullet$;
$p \geq k$	${}^\bullet p$;
$p = k$	$\begin{cases} {}^\bullet p & \text{if } m(p) < k \\ p^\bullet & \text{if } m(p) > k \end{cases}$
$p \neq k$	${}^\bullet p \cup p^\bullet$.

Let $\Phi = \phi_1 \wedge \phi_2$. Since ϕ is not satisfied at m there is an $i \in \{1, 2\}$ such that ϕ_i is not satisfied at m. If none of ϕ_1 and ϕ_2 are satisfied, let i be either of both indices. On any path between the current marking and a marking satisfying ϕ, the truth value of ϕ_i must change from false to true. Thus, a transition from $A_{\phi_i}(m)$ must occur. Consequently, we can set $A_\phi(m) = A_{\phi_i}(m)$. If we express the reachability problem for m^* by the satisfiability problem $\bigwedge_{p \in P} p = m^*(p)$, then we obtain exactly the stubborn sets as discussed for reachability.

Let $\phi = \phi_1 \vee \phi_2$. Since ϕ is not satisfied at m, neither of ϕ_1 and ϕ_2 are satisfied. Thus, between the current marking and a marking satisfying ϕ, the truth value of either ϕ_1 or ϕ_2 must change from false to true. Thus, a transition of $A_{\phi_1}(m)$ or $A_{\phi_2}(m)$ must occur. Therefore, we can set $A_\phi(m) = A_{\phi_1}(m) \cup A_{\phi_2}(m)$.

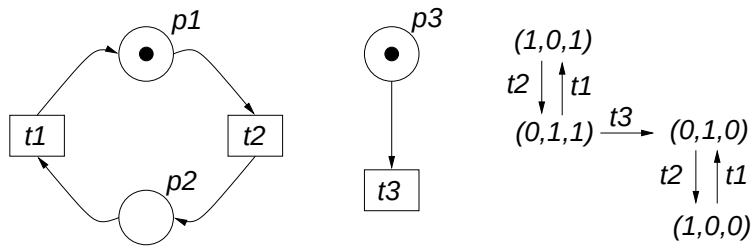


Fig. 5. Reduced graph not preserving shortest paths

Theorem 4 (Preserve satisfiability of state predicates). *Using weakly stubborn supersets of $A_\phi(m)$ at every marking, satisfiability of ϕ is preserved in the reduced graph.*

Corollary 1 (Preserve invariance of state predicates). *Using weakly stubborn supersets of $A_{\neg\phi}(m)$ at every marking, invariance of ϕ is preserved in the reduced graph.*

In this framework, many simple standard Petri net properties can be expressed.

It is difficult to compare our approach with language preserving stubborn sets. However, the stubborn sets are different. Language preserving stubborn sets contain either no enabled visible transition or all visible transitions (i.e. transitions in the environment of places appearing in ϕ). Our stubborn sets contain always at least some visible transitions, but not necessarily all of them. Thus it is likely that language preserving stubborn sets behave better in those cases where many transitions are invisible while our approach is recommendable for problems with many visible transitions. As an example for the total absence of invisible transitions we have already studied the reachability problem.

10 Shortest paths

In many situations, the existence of a path to a state satisfying a predicate is not the only useful piece of information in the context of reachability. For several optimisation problems, the shortest path to a state satisfying the predicate is of interest. For debugging purposes, short witness paths (for satisfiability) and counterexample paths (for invariance) are welcome as well. A shortest path to a state predicate is a transition sequence starting at m_0, leading to a marking that satisfies the predicate, and having minimal length.

Extended, language, LTL, and CTL preserving stubborn sets do not preserve shortest paths.

Consider Figure 5 and the state predicate $p3 = 0$. The only visible transition with respect to this predicate is $t3$. At m_0, $\{t2\}$ is stubborn and $t2$ is invisible. Thus, it is possible to fire only $t2$ at m_0. In the reduced graph (depicted in the same figure), the shortest path to a marking satisfying $p3 = 0$ has length 2. In the original graph, a marking satisfying the predicate can be reached by firing only one transition, namely t3.

Basic stubborn sets do preserve shortest paths to dead states. This fact can be easily deduced from the original proofs but has, to the authors best knowledge, not received the attention it deserves.

Theorem 5 (One shortest path is preserved). *Using our stubborn set approach to the satisfiability problem, a shortest path in the reduced graph to the given predicate ϕ has the same length as a shortest path of the full graph.*

Proof. As the reduced graph is a subgraph of the full one, a shortest path in the reduced graph cannot be shorter than a shortest path in the full graph.

Let l be the length of a shortest path in the original graph to ϕ. We show that there is a path of same length in the reduced graph. Assume there is no path of considered length in the reduced graph. Since we consider paths starting at the initial marking, every path w of length l can be divided into $w = w_1 w_2$ where w_1 appears in the reduced graph as well and w_2 does or does not have arcs in common with the reduced graph. According to our assumption, w_2 is not empty. Select, among all paths from m_0 to ϕ with length l, such w^*, w_1^* and w_2^* that length of w_1^* is maximal, i.e. the path that leaves the reduced subgraph for the first time as late as possible. Let m^* be the marking satisfying $m_0 \xrightarrow[A_\phi]{w_1} m^*$ and consider the stubborn set used at m^*. Since w_2^* is a path from m^* leading to a marking m satisfying ϕ, it contains elements of the stubborn set at m^* (see the preservation theorem for details). Thus, we can split w_2^* into sequences v_1^*, v_2^*, and a transition t^* where $w_2^* = v_1^* t^* v_2^*$, v_1^* does not contain elements of the stubborn set used at m^*, and t^* is an element of the stubborn set used at m^*. According to the definition of stubborn sets it holds $m^* \xrightarrow[A_\phi]{t^*} m_1 \xrightarrow[R]{v_1^* v_2^*} m$. Since t^* is in the stubborn set used at m^*, m_1 appears in the reduced graph. Thus, $w_1^* t^* v_1^* v_2^*$ is a path in the full graph of length l leading from m_0 to m. m satisfies ϕ. Furthermore this sequence can be split into $w_1^* t^*$ and $v_1^* v_2^*$ where the first part is completely contained in the reduced part and longer than w_1^*. This is a contradiction to the assumed maximality of length of w_1^*. □

The proof shows that the reduced graph contains at least one permutation of every shortest sequence to a marking satisfying the predicate.

Corollary 2 (Cheapest paths are preserved). *Let $c : T \longrightarrow \mathbb{Q}^+$ be a cost function. Define the costs of a sequence as the sum of the costs of the occurring transitions. Then our stubborn set approach to the satisfiability of state predicates preserves a cheapest path.*

11 Dead transitions

Dead transitions are preserved by extended stubborn sets. In our approach, stubborn sets in the weak sense can be used. On the other hand, extended stubborn sets preserve *all* dead transitions while our approach requires to specify a single transition to be preserved. However, this setting might be useful in the context of the fact technique.

We obtain stubborn sets that are equivalent to those that can be obtained with tester processes [12]. However, we present the approach in order to show the consistency of our concept for standard properties.

Definition 10 (Dead transition). *A transition is dead (according to a transition system) if there is no m such that $m_0 \xrightarrow{*} m$ and t is enabled at m.*

Theorem 6. *Use, at every m, a stubborn superset (in the weak sense) of $\{t\}$ (i.e. $\{t\}$ is the attractor set) and t will be dead according to the reduced graph iff it is dead according to the full graph. If t is not dead, a shortest sequence from m_0 to a marking enabling t is preserved.*

The proof is straightforward when the common patterns of the previous proofs are used.

12 Dead markings revisited

Our approach does not add anything to the dead marking problem. However, we obtain a slightly different perspective to the problem. Furthermore we can show that our approach is consistent for a large class of properties.

Definition 11 (Dead marking). *A marking is dead iff it is reachable from the initial marking and no transition is enabled at it.*

We construct a reduced graph that contains all dead markings of the full graph. As usual, we identify a set of transitions that attracts graph generation to the desired class of markings. Consider a marking m. If m is not dead then there is an enabled transition t at m. Between m and a dead marking some event must happen that disables t. Thus, t or a conflicting transition must occur. That is, our approach suggests to use stubborn supersets of the attractor set $({}^\bullet t)^\bullet$.

In the original approach, we choose an enabled transition and compute a stubborn superset of it. At the first glance, this seems to yield smaller stubborn sets. However, adding the conflicting transitions to an enabled transition is usually part of the closure operation. Thus, the original method leads to the same stubborn set as ours.[1]

[1] Some optimisations basically exploit the non–determinism for the choice of the starting transition (i.e., they may switch to another starting transition during stubborn set calculation), or they improve upon the treatment of transitions with arcs in both directions to a place. In principle, these optimisations can be applied to our stubborn sets as well.

13 Stubborn sets and terminal components

The component structure of the reachability graph provides valuable information about the system behaviour. Several properties, among them liveness, reversibility, and home states, can be defined in terms of terminal strongly connected components of the reachability graph, at least if the reachability graph is finite. In this section, we consider satisfiability of state predicates in terminal components. Thereby we restrict ourselves to bounded (i.e. finite state) systems. Given a state predicate ϕ, we want to preserve the following situations:

- in all terminal components there is a state satisfying ϕ;
- there is a terminal component where all states satisfy ϕ;

These constellations correspond to temporal expressions such as "it is always possible to satisfy ϕ", and "it is possible to make ϕ hold permanently". These situations could be called liveness of ϕ and (weak) stability of ϕ.

Theorem 7 (Stubborn sets for TSCC based properties). *Consider an arbitrary state predicate ϕ and a reduced transition system constructed using attractor sets A_ϕ as defined in Section 9, using stubborn sets in the strong sense, and avoiding ignored transitions. Then every TSCC of the reduced graph contains a marking satisfying ϕ iff the full graph does.*

Via contraposition and using the negated state predicate, the theorem covers weak stability as well.

Proof.
Part 1
Assumption: Every TSCC of the full graph contains a marking satisfying ϕ.
Claim: Every TSCC of the reduced graph contains a marking satisfying ϕ.

Let C be a TSCC of the reduced graph and $m \in C$. m is contained somewhere in the full graph as well. Thus (by basic properties of SCC) some TSCC of the full graph can be reached from m and inside this TSCC is a state satisfying ϕ. Hence, there is a path in the full graph from m leading to a marking satisfying ϕ. Let m^* be the first marking satisfying ϕ on this path. According to the considerations about preservation of state predicates, some path from m to m^* is contained in the reduced graph as well. Since C is a terminal SCC of the reduced graph, this path cannot leave C. Thus, $m^* \in C$.

Part 2
Assumption: Every TSCC of the reduced graph contains a marking satisfying ϕ.
Claim: Every TSCC of the full graph contains a marking satisfying ϕ.

Let C be a TSCC of the full graph. By construction and Proposition 1, there is a $m \in C$ contained somewhere in the reduced graph as well. Thus, some TSCC of the reduced graph is reachable from m and there a marking m^* satisfying ϕ. Since the reduced transition system is a subgraph of the full graph, the same

path exists in the full graph. Since the path cannot leave the terminal SCC C, m^* is contained in C as well. □

In this generality, our approach complements previous methods. In general, TSCC based properties cannot be expressed in terms of net languages or LTL. Basic stubborn sets sometimes cut off all TSCC. Extended stubborn sets preserve selected properties, but not arbitrary ones. In CTL, all properties can be expressed (via the operators AG and EF) but the CTL–preserving method imposes much stronger restrictions to the reduced graph.

14 Liveness, reversibility

Definition 12 (Liveness, reversibility). *Consider an arbitrary finite transition system. A transition t is live iff every TSCC of the transition system contains a marking where t is enabled. The transition system is reversible iff every TSCC contains m_0 (i.e. there is exactly one TSCC).*

Due to our restriction to finite state systems, our definition of liveness is equivalent to the more popular one: for every reachable state another state is reachable where t is enabled. Reversibility can be directly traced back to results of the previous section. For liveness, we can either use the enabling condition and use results of the previous section, or we can compute a reduced graph without ignored transitions using stubborn supersets of attractor set $\{t\}$.

Extended stubborn sets preserve liveness of all transitions. Since our method imposes all restrictions of extended stubborn sets, the new approach cannot lead to better reduction than extended stubborn sets. Thus, we have a case where the existing methods are strictly more powerful than our one.

To the author's best knowledge, a stubborn set approach to reversibility has not been reported elsewhere. Among the existing general approaches, only CTL can express reversibility, but its stubborn set method fails due to the absence of invisible transitions (unless there are transitions that do not change the marking at all).

15 Home states

Definition 13 (Home state). *If the reachability graph of a Petri net has exactly one TSCC, then its elements are home states. If the reachability graph has more than one TSCC, the net does not have home states.*

We have to distinguish the existence of home states from the verification whether a particular state m is a home state. The second query is equivalent to the appearance of m in every TSCC. Thus, the method of section 13 applies. If

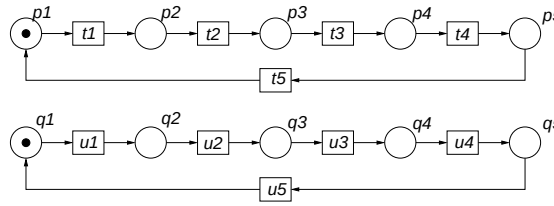


Fig. 6. Net with home states

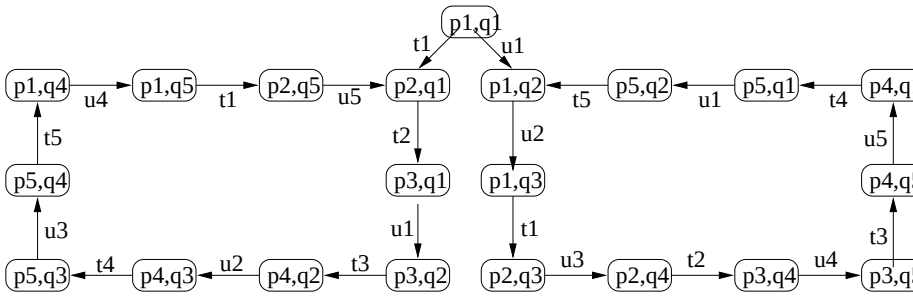


Fig. 7. Reduced graph not exhibiting the home state property

one is interested in all home states, one can use any single home state m and compute the set of all states reachable from m. Thus, it is sufficient to study the existence of home states.

> Due to the existential quantification of states, the existence of home states cannot be expressed in LTL nor in CTL. Language preserving stubborn sets do not help either. As the (reversible) net in figure 6 with the reduced graph in figure 7 demonstrates, extended stubborn sets do not preserve the existence of homes states as well. Intuitively, the reduced graph with respect to extended stubborn sets does not necessarily connect nodes strongly that are strongly connected in the full graph.

For preserving home states, we propose to compute first the reduced graph using extended stubborn sets. Let m_C be any representative of a SCC C in the reduced graph and consider the set $H = \{m_C \mid C \text{ is TSCC of the reduced graph }\}$. By simple properties of SCC, and Proposition 1, this set contains an element of every TSCC of the original graph. If there are m_1 and m_2 in H where $m_1 \xrightarrow[R]{*} m_2$ then m_1 is either member of a non–terminal SCC (in the full graph) or member of the same SCC in the full graph as m_2. Thus, in this case m_1 can be removed from H and H does still contain at least one element of every TSCC of the full graph. Therefore, we check mutual reachability of the elements in H. For this purpose we can use the new stubborn set method for reachability. When the removal process terminates with a singleton set, the net has home states and the unique element of H is one. If the removal process terminates with more

than one element in H, the net does not have home states since the remaining elements must be members of different TSCC of the full graph.

16 Any property missing?

There is a unique pattern behind all solutions presented in this paper. Identify a set of states that you want to preserve. Then, find a set of transitions $A(m)$ (depending or not depending on the current state m) such that every path starting from the current marking and leading to a marking to be preserved contains at least one transition in $A(m)$. If you want to preserve the plain occurrence of desired states, use basic stubborn sets (in the weak sense). If you want to preserve the occurrence of desired states in terminal SCC, use extended stubborn sets.

It is likely that several other properties than those presented in this paper can be preserved this way. So one could ask whether results of this paper can be generalised to one of the popular temporal logics. Since in this paper we always deal with preserving certain *states* rather than *paths*, the logic CTL is a better candidate for such a generalisation than LTL. Moreover, our method depends on situations where there is a difference between the current state and the desired states (which needs to be bridged by the attractor transitions). Therefore, it turned out that we can provide a new methodology to preserve CTL features like EF ("there is a state in the future where...") while we cannot replace the current approach for features like EG ("there is a path where all states satisfy..."). In the latter case, the current state cannot be distinguished from the "good" future states, so we cannot define any reasonable attractor set. However, details about a modified CTL model checking approach are subject of a forthcoming paper.

17 Conclusion

The major difference between many earlier (general) methodologies of stubborn set application and our approach is the meaning of *preservation*. The language preserving [10] as well as the LTL preserving approach [11, 12, 6, 3] preserve *paths*. The CTL approach additionally preserves important parts of the branching behaviour between paths. All these methods try to "hide" the reduction by restricting it to the invisible part of the system behaviour. We preserve *states*. We try to approach these states directly without caring about visibility.

Thus, our approach complements the existing stubborn set toolkit. It is closely related to the tester concept [12], but treats visibility in a different way. We were able to find suitable stubborn sets for a large set of standard properties. Especially for global properties (i.e. when invisible transitions are absent), our approach still keeps the opportunity of significant reduction. If only few transitions are visible, the language preserving method has more room to work.

As a remarkable new feature, we constantly preserve shortest as well as cheapest witness or counterexample paths for reachability, satisfiability, invariance,

and dead transition problems (i.e. properties that can be defined as plain existence of certain states). This makes stubborn set based reduction applicable to some quantitative analysis queries such as scheduling problems.

Acknowlegdements

The author would like to thank Hagen Völzer for asking the right question at the right time. Stephan Roch, Stefan Haar, Kimmo Varpaaniemi, and other colleagues from Humboldt–Universität zu Berlin and Helsinki University of Technology contributed important comments to earlier versions of the paper. The author is grateful for the valuable comments he received through the refereeing process for the present conference.

References

1. A. Finkel. A minimal coverability graph for Petri nets. *Proc. of the 11th Int. Conf. Application and Theory of Petri nets*, pages 1–21, 1990.
2. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *3rd Israel Symp. on the Theory of Computing and Systems, IEEE 1995*, pages 130–140, 1995.
3. P. Godefroid and P. Wolper. A partial approach to model checking. *6th IEEE Symp. on Logic in Computer Science, Amsterdam*, pages 406–415, 1991.
4. R. M. Karp and R. E. Miller. Parallel program schemata. *Journ. Computer and System Sciences 4*, pages 147–195, May 1969.
5. I. Kokkarinen, D. Peled, and A. Valmari. Relaxed visibility enhances partial order reduction. *9th Int. Conf. Computer Aided Verification, LNCS 1254*, pages 328–339, 1997.
6. D. Peled. All from one, one for all: on model–checking using representitives. *5th Int. Conf. Computer Aided Verification, LNCS 697*, pages 409–423, 1993.
7. S. Roch and P. Starke. INA Integrierter Netz Analysator Version 2.1. Technical Report 102, Humboldt–Universität zu Berlin, 1998.
8. P. Starke. *Private communication*, 1998.
9. A. Valmari. Error detection by reduced reachability graph generation. *9th European Workshop on Application and Theory of Petri nets*, pages 95–112, 1988.
10. A. Valmari. Stubborn sets for reduced state space generation. *Advances in Petri Nets 1990, LNCS 483*, pages 491–511, 1991.
11. A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design 1*, pages 297–322, 1992.
12. A. Valmari. On-the-fly verification with stubborn sets. *5th Int. Conf. Computer Aided Verification, LNCS 697*, pages 397–408, 1993.
13. A. Valmari. State of the art report: stubborn sets. *Petri Net Newsletter 46*, pages 6–14, 1994.
14. A. Valmari. Stubborn set methods for process algebras. *Workshop on Partial Order Methods in Verification, Princeton*, pages 192–210, 1996.
15. A. Valmari. The state explosion problem. *Lectures on Petri Nets I: Basic Models, LNCS 1491*, pages 429–528, 1998.
16. K. Varpaaniemi. *Private communication*, 1993.
17. K. Varpaaniemi. *On the stubborn set method in reduced state space generation.* PhD thesis, Helsinki University of Technology, 1998.
18. K. Varpaaniemi. *Private communication*, 1998.

Petri Net Based Behavioural Specification of CORBA Systems

Rémi Bastide, Philippe Palanque, Ousmane Sy, Duc-Hoa Le, and David Navarre

LIHS-FROGIS, Université Toulouse I, Place Anatole France
F-31042 Toulouse CEDEX, France
{bastide, palanque, sy, leduchoa, navarre}@univ-tlse1.fr

Abstract. CORBA is a standard proposed by the Object Management Group (OMG) in order to promote interoperability between distributed object systems. CORBA provides a programming-language neutral Interface Definition Language (IDL) that describes the syntactic aspects of services supported by remote objects. However, CORBA IDL does not provide any means to specify the behaviour of objects in an abstract and formal way. Behaviour specification is provided either in plain English, or directly in the programming language chosen for the implementation. We propose the use of Cooperative Objects, a dialect of object-structured high-level Petri nets, as formalism for behavioural specification of CORBA systems. We detail at the syntactic and semantic level how this formalism supports the features of the CORBA object model. We present a realistic case study to demonstrate our approach.

Keywords: Distributed systems, CORBA, behavioural specification, high-level Petri nets.

1 Introduction

CORBA [16], [21] (Common Object Request Broker Architecture) is a standard proposed by the Object Management Group (OMG) in order to promote interoperability between distributed object systems. The appearance of an industrial standard is an indication that the field of object-oriented distributed computing has moved, in the past few years, from experimental research projects to mainstream commercial products.
CORBA proposes an Interface Definition Language (IDL), independent from any programming language (although closely patterned after C++) and object-oriented, supporting specialisation of interfaces through inheritance. A CORBA-IDL interface specifies at a syntactic level the services that a client object can request from a server object that implements this interface. The interface details the services supported and their signature: a list of parameters with their IDL type and parameter-passing mode, the IDL type of the return value, the exceptions that may possibly be raised during the processing of the service.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 66-85, 1999.
© Springer-Verlag Berlin Heidelberg 1999

```
interface Example {
      exception reject;
      integer op1(in string a);
      void op2(inout float b, out integer c )
            raises(reject);
}
```

Fig. 1. An example of CORBA IDL

Fig. 1 illustrates the definition of an interface in CORBA IDL. This text defines one interface (*Example*) supporting two services (*op1* and *op2*). The figure also illustrates the syntax for the various parameter-passing modes (*in*, *out* or *inout*) and of the exceptions. The keyword *exception* defines an exception type, while the keyword *raises* specifies what types of exceptions may be raised by a service.

A recognised limitation of CORBA is that it defines remote object classes in terms of their interface only. CORBA IDL covers only the syntactic aspects of the possible use of a remote object and does not cover any semantic or behavioural description, while this information is obviously of prime importance for the clients. By behavioural aspect, we mean:

- The constraints on the order of invocation of the services described in the interface.
- The concurrency constraints of the remote object: is it able to support concurrent access to its services, or does it enforce a serialisation on the concurrent invocations?
- The conditions under which an exception might be raised during the processing of a service.

What CORBA lacks is an abstract way to specify the semantics of an IDL interface without constraining its implementation, much in the same way that an Abstract Data Type [8] (ADT) specification can be used for specifying the semantics of a sequential data type.

The present paper aims at providing a suitable solution to the problem of behavioural specification of distributed objects, in the context of CORBA. The paper is organised as follows: We first detail the requirements for a behavioural specification formalism suited to CORBA. We then present how the Cooperative Objects formalism needs to be adapted in order to support fully the CORBA model. Section 3 presents a significant case study of specification using our approach. Section 4 explains how the formalism can be used to enable rapid prototyping of distributed systems.

2 Behavioural Specification of CORBA Systems

Our goal is provide a notation suited to the behavioural specification of CORBA systems: we want to be able to describe the behaviour of a collection of interacting objects, and not merely the behaviour of a single object in isolation. A formal notation aimed at serving this goal has to comply with several requirements:

- Cope with data-flow as well as with control flow. The formalism needs to be able to deal with typed values, and not only with pure causal relationships. It will often be the case that the behaviour of a CORBA system depends not only on the previous history of invocations between objects, but also on the values exchanged during previous invocations. For a given state of an object, an invocation may succeed or fail according to the values of parameters of the invocation. This obligation to deal with data-flow as well as control-flow rules out the use of "basic" Petri nets as a potential formalism, since we need coloured tokens to model the data exchanged between objects.
- Cope with the dynamicity of object references: A behavioural specification formalism for CORBA has to support its fundamental object model, and more precisely to allow remote objects to be designated by references, and these references to be transmitted as invocation parameters. Several object-oriented Petri net formalisms [23], [13], aimed at providing a solid theoretical basis to concurrent object-oriented computation, refrain from using references, and prefer considering tokens as objects, and not references to objects. Such formalism prevent a same object to be referenced by different tokens in different places, and thus would make the specification of a reference-based system such as CORBA less straightforward. The work of [15], which aims at providing Petri-net-based semantics to agent-based systems, deals directly with the dynamics of object references.
- Allow specifying internal concurrency for objects. This point is especially important for CORBA since a CORBA server object will often be a "large grained" entity shared by a lot of clients, providing services whose processing will take some time. It is therefore unrealistic to enforce each service to be atomic, so that at most one service will be active at any time at the server. Actually, all of current CORBA ORBs allow for "multi-threaded" server implementations, where a server object can serve several services at the same time. Some attempts have been made to extend the theory of Abstract Data Types in order to specify the behaviour of concurrent objects [14]. Most of these approaches consider an object as a monitor, allowing only one service to be active at a time, thus cannot be extended to the modelling of distributed CORBA servers with internal concurrency.
- Serve the needs of the implementers of the server class, as well as those of the designers of systems that will be clients for this server. On the one hand, the behavioural specification must be complete and precise enough that the programmer implementing the server in some programming language knows in a precise and non-ambiguous way what behaviour to implement; the specification must be abstract enough not to constraint the implementation choices of the programmer. On the other hand, the potential clients of the class will use the specification to gain a non-ambiguous understanding of the semantics of each service.

The behavioural specification of object systems is a field of research per se [11]. Among the various formalisms proposed, StateCharts [9] comply with the above requirements, and is actually the behavioural formalism for the popular Unified Modelling Language (UML) notation [18], [19]. Several attempts have been made to use process algebraic techniques in this domain, notably the work of [7], which provides an algebraic semantics to the various invocation modes of CORBA.

We propose the use of high-level Petri nets as the behavioural specification medium for CORBA systems. The usefulness of combining Petri nets and objects has been recognised by several authors, as witnessed by the workshops held at previous editions of the ATPN conference [1], [2]. More particularly, we use an object-oriented dialect of high-level Petri nets called Cooperative Objects. This formalism complies with the requirement listed above, and its Petri net roots provide is with a powerful basis for modelling and analysing concurrent behaviours.

2.1 The Cooperative Object Formalism

Cooperative Objects [5] (CO) are a dialect of object-structured, high-level Petri nets. Their lengthy formal definition has been provided in previous publications [3], [20], and we will only recall informally their main features, through examples. CO can be considered as a dialect of coloured Petri nets [10], of which they differ mainly by their object-oriented features and by the different arc inscriptions.

A CO class specifies a class of objects by providing their interface (the set of services offered, along with their signature) and their dynamic behaviour. The behaviour of a CO class is called its *Object Control Structure* (ObCS), and is defined with a dialect of high-level Petri nets. More specifically:

- Tokens are tuples of typed values. The arity of a token is the number of values it holds, and tokens of zero-arity are thus the "basic" tokens used in conventional Petri nets. We will call *Token-type* a tuple of types, describing the individual types of the values held by a token. Token-types will be noted <$Type_1$, … $Type_n$> or just <> to denote the Token-type of zero-arity tokens.
- Places are defined to hold tokens of a certain Token-type; thus all tokens stored in one place have the same Token-type and arity. A place holds a multiset of tokens; thus a given token may be present several times in the same place.
- Each arc is inscribed by a tuple of variables, with a given multiplicity. The arity of an arc is the number of variables associated to it. The arity of an arc is necessarily the same as the arity of the Token-type of the place it is connected to, and the type of each variable is deduced from this Token-type. The multiplicity of an arc is the number of identical tokens that will be processed by the firing of a transition associated to this arc. The general form of an arc inscription is `multiplicity*<v1,… vn>`. A multiplicity of 1 can be omitted (thus `1*<v1,… vn>` can be abbreviated as `<v1,… vn>`) and an empty list of variables can also be omitted (thus `2*<>` can be abbreviated as `2`).
- Transitions have a precondition (a Boolean expression of their input variables) and an action, which may use any service allowed for the types of their input or output

variables. The scope and type of each variable of an arc is local to the transition the arc connects to.

A transition is enabled when:

- A substitution of its input variables to values stored in the tokens of its input places can be found
- The multiplicity of each substituted token in the input places is superior or equal to the multiplicity of the input arc,
- The precondition of the transition evaluates to true for the substitution.

The firing of a transition will execute the transition's action, compute new tokens and store them in the output places of the transition. The formalism also supports two arc extensions [12]: test arcs and generalised inhibitor arcs.

```
place p1 <string, integer> =
      { 3*<"hello",1>, 2*<"hello",2>, <"bye",3> };
place p2 <integer> = { <1>, <3> };
place p3 <integer> = { <0>, <3>, <4> };
place p4 <integer> = { };
transition T1 {
      precondition { true };
      action {};
}
transition T2 {
      precondition { b > 0 };
      action { c = b+1; }
}
```

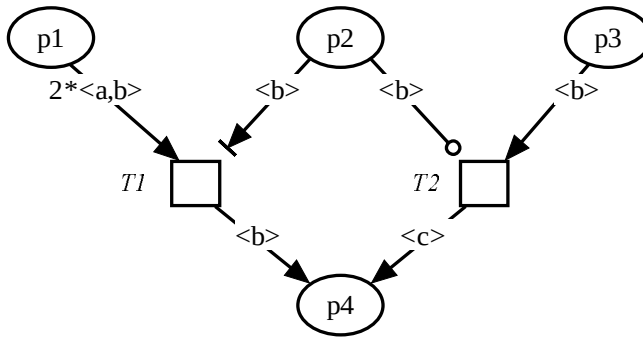


Fig. 2. Excerpt of a CO class

Fig. 2 shows an excerpt of a CO net, along with the textual notation used to describe the Token-types of the places, their initial marking and the transition's precondition and action.

The arc between *p2* and *T1* is a test arc, and the arc between *p2* and *T2* is a generalised inhibitor arc. The *T1* transition is only enabled by the following

substitutions: {a => “hello”, b => 1}. It is not enabled by {a => “hello”, b => 2} since no substitution with b = 2 can be found in place *p2*. It is not enabled by {a => “bye”, b => 3} since the token <”bye”, 3> has insufficient multiplicity in place *p1*.

Likewise, *T2* is only enabled by { b => 4 }. The inhibitor arc between *p2* and *T2* prevents the substitution { b => 3 }, while the precondition of *T2* prevents the substitution { b => 0 }.

From the initial marking described in Fig. 2, the occurrence of *T1* would result in a marking of { 1*<”hello”,1>, 2*<”hello”,2>, <”bye”,3> } in *p1*, and of { <1> } in *p4*, the markings of *p2* and *p3* remaining unchanged. From the same initial marking, the occurrence of *T2* would result in a marking of { <0>, <3> } in *p3* and of { <5> } in *p4*, the marking of *p1* and *p2* remaining unchanged.

2.2 Integrating CORBA IDL and Cooperative Objects

Cooperative Objects were initially defined independently of CORBA [3], but CO and CORBA happen to share the same object model, as described in §1. CO and CORBA complement each other nicely: The initial description of CO used an idiosyncratic language to describe interfaces, while CORBA provides an attractive, language neutral IDL. On the other hand, CO can be used to provide the behavioural information lacking in CORBA-IDL.
This section presents the necessary syntactic adaptation that CO require to match more closely CORBA-IDL, and some additions to the formalism supporting CORBA-specific constructs such as exceptions and parameter-passing modes.

CORBA-IDL will be used as the data description language used by Cooperative Objects. Cooperative Objects will use IDL to describe:

- The system of data types used by the ObCS nets of Cooperative Objects. The Token-type of the places will thus be described in terms of CORBA IDL, and the variables on the arc will be of an IDL-defined type.
- The interface of a Cooperative Object class itself.

To achieve the integration of Cooperative Objects and CORBA IDL, we need to define a mapping from the constructs of CORBA IDL to those of Cooperative Objects, much in the same way that CORBA defines mappings from IDL to conventional programming languages such as C++, Smalltalk or Java.

We now illustrate the CO-CORBA integration by showing how a CO class can add to the interface of Fig. 1 the behavioural aspects that are needed to complete its definition.

Fig. 3 shows a CO class definition, corresponding to the CORBA-IDL of Fig. 1. Part of this class definition is directly deduced from the interface, while the other part corresponds to the behavioural specification added by the designer. The parts added are greyed out for clarity.

```
class ExampleSpec specifies Example {
      place op1 <string>;
      place op1 <integer>;
      place op2 <float>;
      place op2 <float, integer>;
      place op2 <Exception>;
      place p1  <string>;
      place p2  <integer>;
      transition T1 {
            action {
                  string x = a.toString();
                  integer y = a;
            }
      }
      transition T2 {
            precondition { b < 0.0; }
      }
      transition T3 {
            precondition { b >= 0.0; }
      }
      // Several place and transition definitions omitted
```

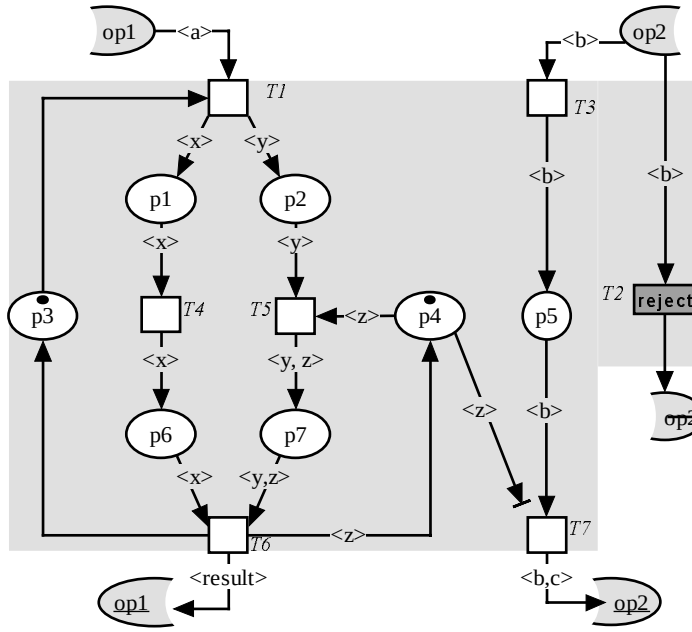


Fig. 3. A CO specification of the Example interface

2.2.1 IDL interfaces and CO Classes

A CO class may specify one or several IDL interfaces. This is convenient since, very often, several interfaces are given for the same entity to allow providing different views of the same object, tailored for the needs of different clients. The CO class of Fig. 3 specifies only one interface, namely *Example*. The keyword **specifies** is followed by the list of CORBA-IDL interfaces specified by the CO class.

2.2.2 Mapping for Services

Each service *op* defined in an IDL interface is mapped to two or three places in the ObCS net: a Service Input Port (SIP, labelled op), a Service Output Port (SOP, labelled op) and a Service Exception port (SEP, labelled ~~op~~), only present if the service may raise an exception. These three places are derived from the IDL, as follows:

- The Token-type of the SIP is the concatenation of the IDL types of all *in* and *inout* parameters of the service;
- The Token-type of the SOP is the concatenation of:
  - the IDL type of the result returned by the service (if any),
  - the list of the IDL types of all *out* and *inout* parameters of the service.
- The Token-type of the SEP is `<Exception>`, where Exception is the super-type of all IDL exception types. The SEP is only used if the service raises an exception.

According to the IDL of Fig. 1, the service op1 is mapped onto two places: op1 for the SIP and op1 for the SOP; op1 has no SEP. Service op2 is mapped onto three places, op2, op2, ~~op2~~. Their Token-types are as follows *op1* : `<string>`, *op1* : `<integer>`, *op2* : `<float>`, *op2* : `<float, integer>`. As service op2 raises the *reject* exception, a SEP ~~op2~~ is added, with Token-type `<Exception>`.

The invocation of one service op results in one token holding all *in* and *inout* parameters being deposited in its SIP. The role of the ObCS net is to process this parameter token in some way, and eventually deposit a result token (holding the result of the service, plus all *out* or *inout* parameters) in the SOP, thus completing the processing of the invocation. An invocation that raises an exception at some point will instead result in an exception token being deposited in the SEP.

In order for the invocation to proceed in a sound way, the ObCS structure must respect a set of constraints. Informally, an object will provide either a result or an exception for each invocation, and will only provide results if it has been previously invoked. With respect to the ObCS, the arrival of one token in the SIP will eventually result later on in exactly one token being deposited either in the SOP or in the SEP.

The necessary and sufficient structural constraints on the ObCS net are as follows:

- **Constraint 1**: The SIP can only have output arcs in the ObCS net;
- **Constraint 2**: The SOP can only have input arcs;
- **Constraint 3**: The SEP can only have input arcs, these arcs coming from specific exception transitions (see 2.2.5).

The ObCS models an "open" server, where the environment (clients) can deposit tokens in SIP and take tokens from SOP or SEP. As we are only considering the server-side, we need to "close" the server system by modifying its ObCS, if we want to investigate its behaviour in isolation.
For each service op, two closing transitions T_{op} and T_{op} are added, and connected to the ObCS in the following way: $Pre(T_{op}) = \{\underline{op}\}$, $Post(T_{op}) = \{op\}$, $Pre(T_{op}) = \{\sout{op}\}$, $Post(T_{op}) = \{op\}$. We define the Operation Control Structure (OpCS) of a service as the set of minimal P-Invariants [6] where SIP, SOP and SEP appear together. Note that the added transitions exist only for computing the P-Invariants, and are not actually part of the ObCS. The P-Invariants are calculated on the ObCS's *underlying net*, i.e. a net where variables are removed from the arcs and token-types are ignored.

- **Constraint 4**: An OpCS has a correct structure if it is not empty (at least one P-Invariant must exist in the ObCS where the Input, Output and Exception port appear together).

With respect to the ObCS presented in Fig. 3, the OpCS of *op1* consists of the places {*op1, p1, p2, p6, p7,* $\underline{op1}$}, since the minimal P-invariants where *op1* and $\underline{op1}$ appear together are {*op1*, *p1, p6,* $\underline{op1}$} and {*op1, p2, p7, op1*}. Likewise, the OpCS of service op2 is {*op2*, *p5*, $\underline{op2}$, $\sout{op2}$}.

2.2.3 Mapping for Parameter-Passing Modes

The semantics for the three parameter-passing modes of CORBA IDL is clear.

- *in* parameters are values provided by the caller, that the service may use at its own will;
- *out* parameters are values computed by the service, and returned to the caller;
- *inout* parameters are values transformed by the service, i.e. provided by the caller and returned to it after being processed.

Constraint 5: Only inout parameters require a special treatment in the OpCS structure: in order to ensure that any inout parameter is properly transmitted from the Service Input Port to the Service Output Port, a sufficient condition is to check that the parameter name appears on every arc connected to each place of the OpCS.

This is the case for service op2 in Fig. 3, as the inout parameter b is transmitted along the P-Invariant {*op2*, *p5*, $\underline{op2}$, $\sout{op2}$}.

2.2.4 Semantics of Invocations in Transitions

We want to be able to model CORBA systems, and not only isolated CORBA servers. Actually, the behaviour of one class in isolation will often be of little interest, and will only become meaningful when we can describe how an instance of the class may interact with other instances of other classes in the system.

CORBA objects interact with one another by invoking services defined by the IDL interfaces they support. The Cooperative Object formalism supports this form of cooperation by allowing the action of a transition to be the invocation of a service.

The operational semantics of such an invocation transition is a client / server protocol that can be formally defined in terms of Petri nets.

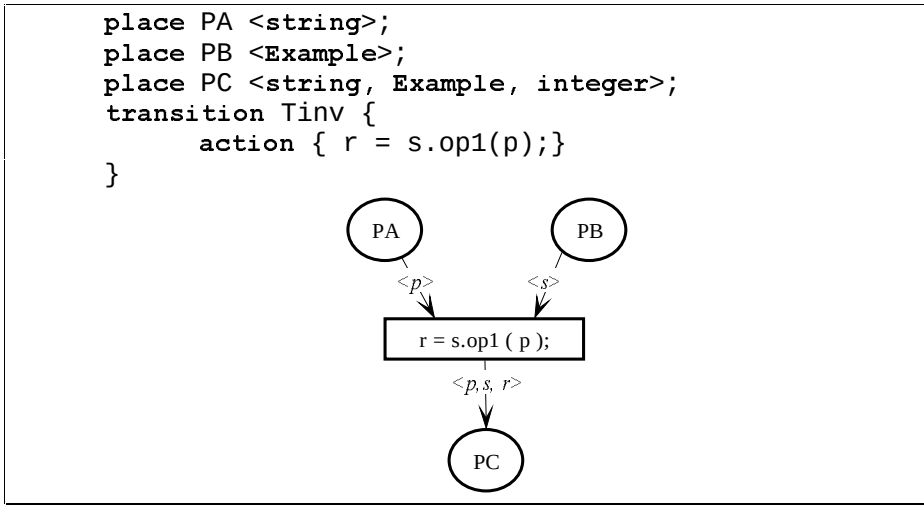


Fig. 4. An example of invocation transition

The client server protocol we use has been first presented in [17] for basic Petri nets, extended to object-oriented high-level Petri nets in [3], and is presented with its full theoretical details in [20]; The fact that not only the internal behaviour of objects, but also their communication protocol is defined in terms of Petri nets enables us to reason about systems of cooperating objects, and not only on isolated instances.

The client-server protocol provides semantics for *invocation transitions* such as the one illustrated in Fig. 4. An invocation transition is a transition whose action is the call of a service supported by one of its input parameters. In the example above, variable *s* is of type *Example*, and the *Example* interface supports the service *op1*.

The semantics of an invocation transition is illustrated in Fig. 5. This semantics requires adaptation of the ObCS on the client side, and on the server side.

On the client's side the adaptation is as follows:

- The invocation transition is considered as a macro-transition extending from the *request transition* to the *complete transition*. The *request transition* constructs a parameter token, including the original parameters of the service and a globally unique call-identifier. The call-identifier is of type CallID. This token is deposited in the *Invocation Parameter port*.
- A *waiting place* is introduced between the request transition and the complete transition. The presence of a token in this place indicates that a call is in progress.
- The results from the service call will be returned to the client in its *Invocation Result Port*. The arrival of a return token will enable the *complete transition*, and terminate the service call on the client's side. It is important to note that the variable *id* is present on both input arcs of the *complete transition*: the transition is only enabled if a substitution is possible between the token values held in the *Waiting* and *Result Port* places, meaning that the same id is found in both tokens. This construct is

necessary to allow a client to issue concurrently several invocations, and to enable the client to match the results it receives with the parameters it has initially provided.

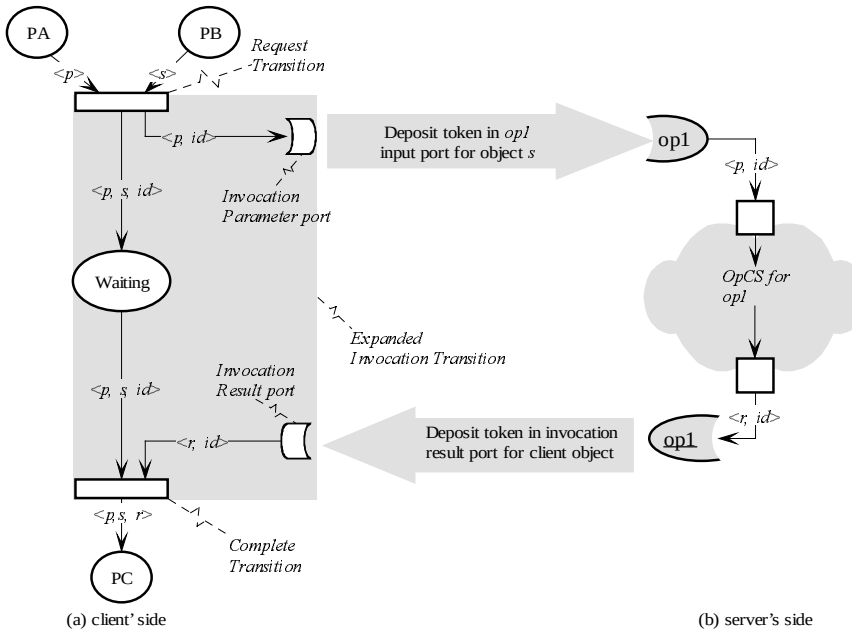


Fig. 5. Semantics of invocation transitions

On the server's side the adaptation is as follows:

The structure of the net is not altered, but only the definition of the places' type and the inscriptions on the arcs. The only requirement for the server is to transmit the call-id within the service subnet, so that the results of the service can be properly routed back to the caller. This is done by appending the type CallID to each place of the service OpCS, and correspondingly adding a variable on the arcs.

2.2.5 Mapping for Exceptions

CORBA IDL allows specifying exceptions that may be raised during the processing of an invocation. An exception is an object of a specific data-type, and can hold information on the causes of its occurrence or other useful data.

When an exception is raised, the normal processing of the service is cancelled, the result, out and inout parameters of the service are undefined, an exception object is instantiated and only this object is transmitted to the client of the invocation.

In order to specify properly the behaviour of a CORBA system, our formalism needs to address two concerns:

- Define under which conditions an exception may be raised during the processing of an invocation, and what corrective actions are eventually needed in the server object to restore a consistent state.
- Define what action a client object needs to take if a service invocation results in an exception instead of providing the expected result.

The first point is tackled by *exception transitions*:

Constraint 6: exception transitions are labelled by the name of the exception data-type that is raised. They can have input and output arcs from any place of the OpCS of one service, but necessarily have exactly one output arc connected to the SEP of this service. The occurrence of an exception transition models the fact that an exceptional condition has occurred during the processing of an invocation, and that this processing cannot be carried any further.

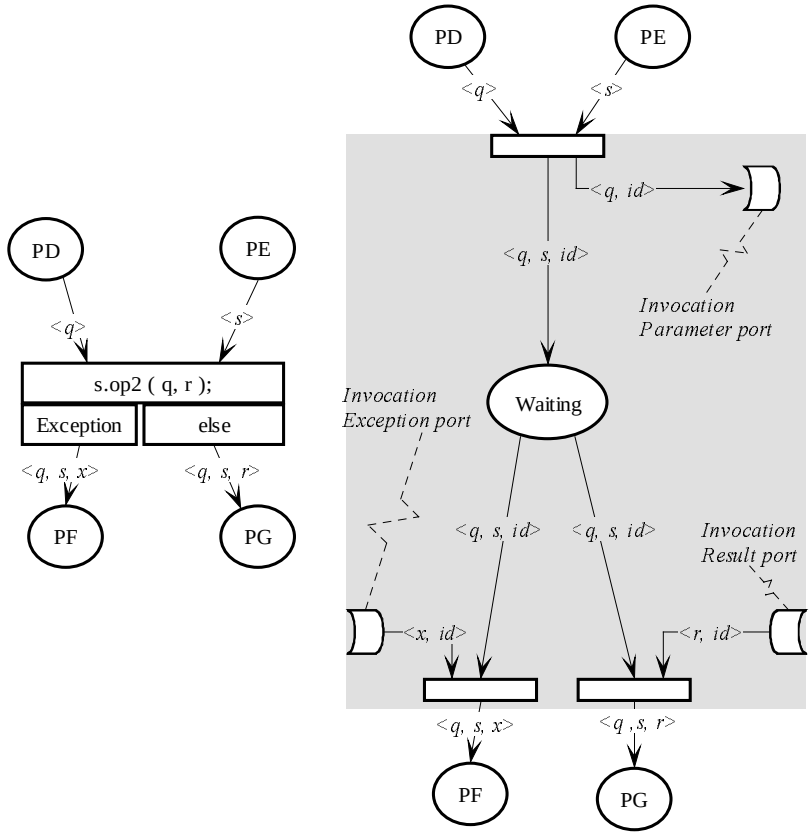


Fig. 6. Invocation transition with exception handling

The *T2* transition in Fig. 3 is an exception transition. It models the fact that the b parameter of the op2 service needs to respect some constraints in order to be properly

processed by the service. In the figure, the T2 and T3 transitions are in structural conflict, but this conflict is solved deterministically since the preconditions of these two transitions are mutually exclusive.

The second point is tackled by:

- a simple syntactic extension of the graphic syntax of invocation transitions (called emission rules),
- a straightforward extension of the client-server protocol described in §2.2.4, acknowledging the fact that an invocation can have two different outcomes: a normal outcome, providing the expected results, or an exception outcome, providing no result other than the exception raised by the server.

Fig. 6 illustrates the graphic syntax of an invocation transition with exception handling (left side) and the associated semantics expressed as a macro-transition (right side). An invocation transition may feature an exception outcome (labelled Exception) and a normal outcome (labelled else). An outgoing arc can only be connected to one outcome. Arcs connected to the exception outcome are labelled by the input variables of the transition, plus a variable to denote the exception object received; they cannot refer to the result of the invocation neither to out or inout parameters.

3 A Case Study in CORBA Behavioural Specification

```
module Banking {
 exception noSuchAccount;
 exception nameAlreadyExists;
 exception insufficientFunds;
 exception accountIsClosed;
 interface Bank {
      Account newAccount(in string name,
            in float initialAmount )
            raises(nameAlreadyExists);
      Account findAccount(in string name )
            raises(noSuchAccount);
      void closeAccount(in Account account)
            raises(noSuchAccount);
 }
 interface Account {
      void transfer( in float amount )
            raises(accountIsClosed, insufficientFunds);
      float balance( );
 }
 interface BankAccount : Account {
      void open();
      void close();
 }
}
```

Fig. 7. The IDL text for the banking system

To illustrate our modelling approach, we now detail an example dealing with the banking business.This example is typical of the intended use of the CORBA technology (although much shorter than real-life examples) and is in fact used as a tutorial example for several commercial ORBs. Despite its small size, it illustrates all the main aspects of our technique, including exceptions, interface inheritance and realistic behavioural specification featuring intra-object concurrency.

The banking system is composed of two different kinds of entities: *bank accounts*, which keep a (positive or null) amount of money, and *banks*, which keep a set of bank accounts. This system is first described in terms of CORBA IDL. Cooperative Object class specifications are then provided, to specify the behaviour of the system.

Fig. 7 shows the interface of the banking system, expressed in CORBA IDL. This IDL first defines data types for the exceptions that might be thrown during processing (*noSuchAccount, nameAlreadyExists, insufficientFunds, accountIsClosed*), and defines three interfaces:

- *Bank*, which the customers will use to create, close, and access their accounts. A bank is merely a repository for bank accounts, and allows associating an *Account* reference to a human-readable name. The service *findAccount* allows retrieving an *Account* reference for which only the name is known;
- *Account*, offering a service to obtain the current balance (balance) and a single service to credit or debit the account (transfer).
- *BankAccount*, which will be used internally by the bank to perform *open* and *close* operations on accounts. *BankAccount* is a specialisation of the *Account* interface, meaning that is offers all the services of *Account*, in addition to the new ones it introduces (*open* and *close*).

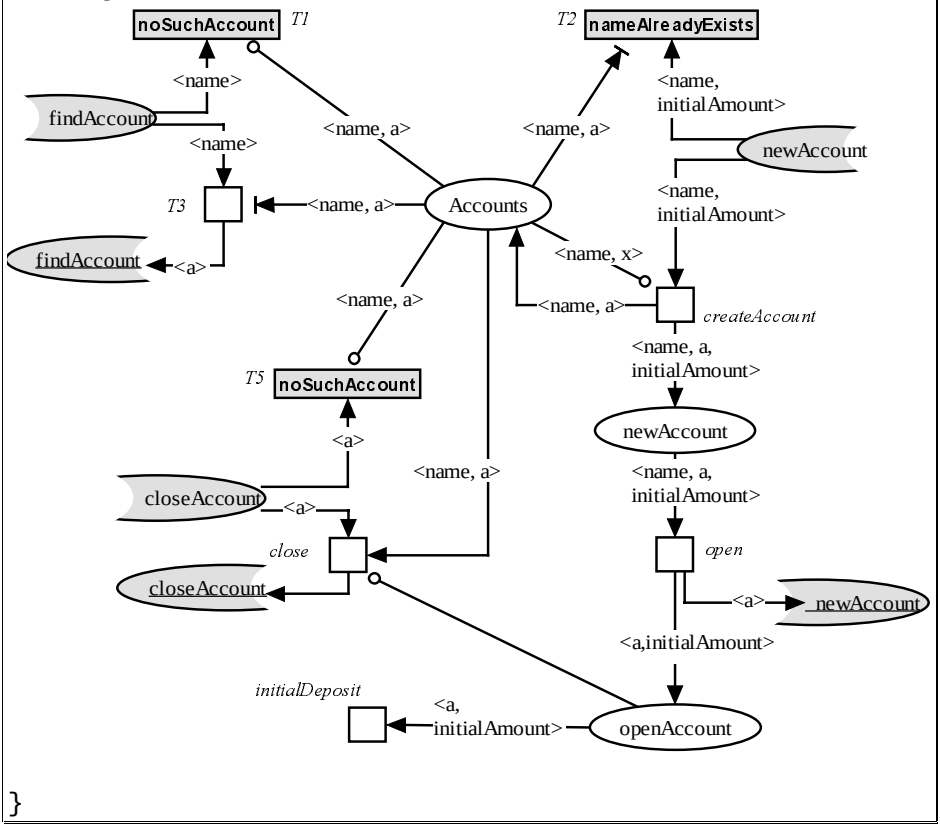
The provision for two different interfaces (*Account* and *BankAccount*) to describe the same concept of account enables to describe different access rights (or different views) of the same object, tailored for the needs of different clients. The signature of the services provided by the *Bank* interface only deal with *Account* references, which means that clients of a *Bank* object will only receive references of this type, and thus will be able to access only the services defined in *Account*.

Obviously a lot more needs to be said, in addition to these interfaces, to have a complete specification of the banking system. The behavioural specification of the banking system is provided below, in terms of two Cooperative Object classes, *BankSpec* and *BankAccountSpec*.

Fig. 8 illustrates both the graphic syntax of a Cooperative Object class and the textual annotations that are necessary to complete its description. These textual annotations are:

- The list of the interfaces that the CO class specifies (keyword **specifies**). In this case, the *BankSpec* class specifies the *Bank* interface.

```
class BankSpec specifies Bank {
      place Accounts <string, BankAccount>;
      place newAccount <string, BankAccount, float>;
      place openAccount <BankAccount, float >;
      transition createAccount {
            action {a = new bankAccountSpec();}
      }
      transition open {
            action { a.open(); }
      }
      transition initialDeposit {
            action { a.transfer(initialAmount); }
      }
      transition close {
            action { a.close(); }
      }
```

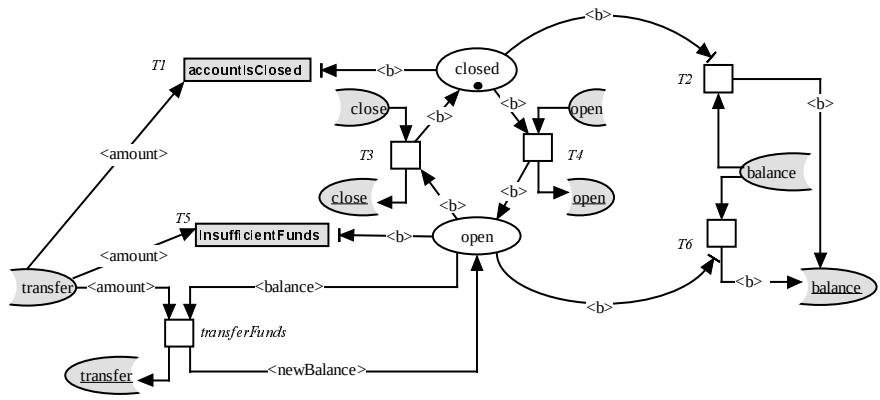



```
}
```

Fig. 8. Cooperative Object class specifying the Bank interface

- The description of the places' type: for instance, the *Accounts* place holds tuples of the form <string, BankAccount>, i.e. the name of a bank account, and the associated reference to a *BankAccount* object. The type for the input and output ports needs not be stated, since it is deduced from the signature of the associated service. For example, the type of the *newAccount* input port is <string, float>, and the type of its output port is <Account>.
- The description of the transitions' preconditions and actions: Empty actions default to no action, and empty preconditions default to true. Only the transitions with non-default precondition or action need to be stated in the textual part.

```
class BankAccountSpec
specifies BankAccount {
      place closed <float> = { <0> };
      place open <float>;
      transition transferFunds {
            precondition {
                  (amount + balance) >= 0;
            }
            action {
                  newBalance = amount + balance;
            }
      }
      transition t5 {
            precondition {
                  (amount + balance) < 0;
            }
      }
```




```
}
```

Fig. 9. CO class specifying the BankAccount interface

The BankSpec class describes a sensible behaviour for the bank entity. For example, it states that an Account reference can be retrieved (through the *findAccount* service) only after an account with the same name has been created (through the

newAccount service) and before it has been closed (through the *closeAccount* service). It also specifies under which conditions exception might be raised (for instance a *nameAlreadyExists* exception is raised when the *newAccount* service is called with a name parameter that matches a previously registered name). Such behavioural constraints could have been modelled just as well using a conventional ADT-based specification, using logical pre and post-conditions for the services. However, the description of more subtle behaviours requires a formalism expressive enough to encompass concurrency and synchronisation constraints.

In this respect, the Operation Control Structure (OpCS) of the *newAccount* service is especially noticeable: unlike the other two services (*findAccount* and *closeAccount*), it is not made of a single atomic transition, but of a subnet encompassing the *createAccount* and *open* transitions. This feature allows for internal concurrency within a *BankSpec* object: While an invocation of *newAccount* is being processed (i.e. when the *newAccount* place holds one token), other incoming invocations of *findAccount* or *closeAccount* can be serviced (if the marking of the *Account* place permits it). Moreover, several invocations of *newAccount* can be serviced concurrently (this will result in the *newAccount* place holding several tokens). Finally, when the service invocation returns (by setting the reference to the newly created *BankAccountSpec* object in the *newAccount* return port), the *BankSpec* object continues an internal processing, namely to initialise the *BankAccountSpec* object with its initial balance (transition *initialDeposit*). This faithfully models the way actual banks proceed in the creation of new customer accounts: If one, as a customer, goes to a bank to create an account with an initial deposit, the account will be created immediately, but it will take some time before the account is actually credited with the initial deposit. If the account's balance is accessed in the meantime, it will be zero. The BankSpec also takes care that an account is not closed before the initial deposit is performed (inhibitor arc between *openAccount* and *close*).

The behavioural specification of bank accounts is shown in Fig. 9. This CO class specifies the *BankAccount* interface and implicitly the *Account* interface since the two are related through inheritance. Several new syntactic constructs are illustrated in this class: Places can be provided with an initial marking, stating their content right after an instance of this class is created. In this case, an account is created in the closed state, with an initial balance of 0, as stated by the definition of the *closed* place. The *transferFunds* and *T5* transitions are in structural conflict. This conflict is deterministically solved by the preconditions of these two transitions, which are mutually exclusive. Thus, the *InsufficientFunds* exception is raised when the current balance is lower than the amount that one tries to withdraw.

Note that (for the sake of brevity), the specification detailed in the *BankSpec* and *BankAccountSpec* classes offers no provision for reopening a closed account. This could be achieved, for instance, by providing a subclass of *BankSpec* with additional services to transfer the balance from a closed account to a newly created account.

4 Prototyping CORBA Systems

A well-known advantage of Petri nets is their executability. This is highly beneficial to our approach, since as soon as a behavioural specification is provided in terms of CO classes, this specification can be interpreted to provide additional insights on the possible evolutions of the system.
Our approach is supported by a tool called PetShop[22], which includes a distributed implementation of the HLPN interpreter described in [4], rewritten in Java. The implementation is such that: 1) An interpreted ObCS can invoke (from an invocation transition) a service of a "third party" CORBA server running outside of the environment; 2) Conversely, an external CORBA client can call a service of a Cooperative Object, regardless of the fact that this service invocation will be performed by interpretation of the CO ObCS. This offers complete interoperability between the CORBA world and our formalism, and enables us to work in realistic settings, where we possess the complete behavioural specification of some objects only, but where we can nonetheless access objects provided by other sources, of which we know only the CORBA IDL.

5 Conclusion and Future Work

The approach presented here is motivated by the momentum gained by CORBA as a standard for distributed object systems, and by the evidence that some form of abstract behavioural modelling supporting the CORBA object model can be of great help in the development life-cycle of such systems.

The main rationale for the use of Petri nets in our approach is their expressiveness in describing complex behaviours, including concurrency and synchronisation. Our belief is that high-level Petri nets are a very efficient behavioural specification language in the context of CORBA, providing an concise, accurate and formal description of the allowed behaviour of objects. Such a description is highly beneficial both to the developers of CORBA servers (which then have a complete and non-ambiguous description of the behaviour they need to implement) and to the programmers of systems that use these servers (for whom the Petri nets serves as the "operating instructions" of the server).

Another well-known advantage of Petri nets is their potential for formal analysis. Our approach has a strong operational bias: we accept to trade full analysis possibilities in favour of modelling power. However, we do not give up the potential for analysis of Petri nets altogether. Currently, our tool includes the usual Petri net analysis algorithms (such as P and T-Invariants, liveness, boundedness, siphons and traps) that operate on the ObCS *underlying net*, i.e. the ObCS where all variables and data types are removed. We are currently in the process of assessing which of the analysis result for basic Petri net are preserved in the high-level ObCS. Our initial studies appear to demonstrate that analysis of the underlying net provides very useful results for the ObCS: for instance, if the underlying net is bounded, then the high-level net is bounded. If the underlying net is unbounded, we have no indication on the boundedness of the ObCS itself, since the boundedness of the high-level net may

result from arc inscriptions or preconditions that are lost in the underlying net. Conversely, if the underlying net is not live, then neither is the ObCS (the reverse implication does not hold). Such properties, although they do not provide definitive answers on the behaviour of the object, are nonetheless of great help to the designer when she is constructing the ObCS. We are also investigating analysis techniques especially devised for high-level nets:

- Analysis of object-oriented features: we wish to use Petri net analysis techniques not only to prove properties on an isolated object, but also to analyse constructs specific to object-oriented systems. For example, when two IDL interfaces are related through inheritance, some form of behavioural inheritance needs to be respected for CO classes that specify these interfaces. The work presented in [24] is a useful starting point for us, the most relevant notion in the context of CORBA appearing to be the one based on the hiding of new methods introduced in subclasses. We also want to analyse the cooperation between several CO instances, to check properties of a system of interacting objects. This is possible since the client-server protocol of §2.2.4 is described in terms of Petri nets, which enables generating a single static Petri net from the ObCS of classes in a system [3], [20]. This global net can be used to prove properties of the system as a whole.
- Test generation: Another ongoing research is the ability to generate test suites from the Cooperative Object class definition. The tool supporting our approach can generate prototypes that are “almost functional” implementations of CORBA-IDL interfaces. They are “almost functional” in the sense that only behavioural requirements are dealt with. Other “quality-of-service” requirements, such as performance, persistence, replication or fault-tolerance, have to be taken care of in a fully functional implementation. The idea is that the CO class can serve as the formal specification of the class, and that programmers will implement the specification in some programming language, implementing the other quality-of-service requirements. This implementation needs to be tested for conformance against the original CO-based specification and the CO specification can be used to generate a test suite for the implementation.

Acknowledgements

The work of Ousmane Sy is funded by CNET, the National Research Centre for the French telecommunications agency (France Telecom) under grant number 98 1B 059. The final goal of the project is to provide a fully integrated CASE tool supporting our approach of behavioural specification of distributed object systems using high-level Petri nets.

The work of Duc-Hoa Le and David Navarre is funded by ESPRIT Reactive LTR project n° 24963, MEFISTO.

References

1. Agha, Gul, and De Cindio, Fiorella. "Workshop on Object-Oriented Programming and Models of Concurrency." *ICATPN'95*, Torino, Italy, June 26-30, 1995. (1995)
2. Agha, Gul, De Cindio, Fiorella, and Yonezawa, Akinori. "2nd International Workshop on Object-Oriented Programming and Models of Concurrency." *ICATPN'96*, Osaka, Japan, June 24, 1996. (1996)
3. Bastide, Rémi. "Objets Coopératifs : Un Formalisme Pour La Modélisation Des Systèmes Concurrents." Ph.D. thesis, Université Toulouse III (1992).
4. Bastide, Rémi, and Palanque, Philippe. "A Petri-Net Based Environment for the Design of Event-Driven Interfaces." *ICATPN'95*.LNCS no. 935. Springer (1995) 66-83.
5. Bastide, Rémi, and Palanque, Philippe. "Modeling a Groupware Editing Tool With Cooperative Objects." *Concurrent Object-Oriented Programming and Petri Nets.* Gul A. Agha, and Fiorella De Cindio, editors. Wien: Springer-Verlag (1998)
6. Desel, Jörg, and Javier Esparza. *Free Choice Petri Nets*. Cambridge Tracts on Computer Science, 40 Cambridge University Press (1995).
7. Gaspari, Mauro, and Zavattaro, Gianluigi. "A Process Algebraic Specification of the New Asynchronous CORBA Messaging Service." *ECOOP'99* (1999)
8. Goguen, J. A., Thatcher, J. W., and Wagner, E. G. "Current Trends in Programming Methodology." *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types.*Prentice-Hall (1978) 80-149.
9. Harel, David, and Gery, Eran. "Executable Object Modeling With Statecharts." *IEEE Computer* 30, no. 7 (1997) 31-42.
10. Jensen, Kurt. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. 2nd edition ed., Vol. 2 Springer-Verlag (1996).
11. Kilov, Haïm, and William Harvey, editors. *Object-Oriented Behavioral Specifications* Kluwer Academic Publishers (1996).
12. Lakos, Charles. "A General Systematic Approach to Arc Extensions for Coloured Petri Nets." *ICATPN'94*, June 1994. LNCS no. 815. Springer (1994) 338-57.
13. Lakos, Charles, and Keen, C. D. "LOOPN++: a New Language for Object-Oriented Petri Nets." *European Simulation Multiconference*, Barcelona, Spain, June 1994. (1994)
14. Meyer, Bertrand. "Systematic Concurrent Object-Oriented Programming." *Communications of the ACM* 36, no. 9 (1993) 56-80.
15. Moldt, Daniel, and Wienberg, Franck. "Multi-Agent-Systems Based on Coloured Petri Nets." *ICATPN'97*, Toulouse France, June 1997. LNCS no. 1248. Springer (1997) 82-101.
16. Object Management Group. *The Common Object Request Broker: Architecture and Specification. CORBA IIOP 2.2 /98-02-01*, Framingham, MA (1998).
17. Ramamoorthy, C. V., and Ho, G. S. "Performance Evaluation of Asynchronous Concurrent Systems." *IEEE Transactions of Software Engineering* 6, no. 5 (1980) 440-449.
18. Rational Software Corporation. *UML Notation Guide*. 1.1 ed.1997.
19. ———. *UML Semantics*. 1.1 ed.1997.
20. Sibertin-Blanc, Christophe. "Cooperative Nets." *ICATPN'94*. LNCS 815. Springer (1994)
21. Siegel, Jon. "OMG Overview: CORBA and the OMA in Enterprise Computing." *Communications of the ACM* 41, no. 10 (1998) 37-43.
22. Sy, Ousmane, Bastide, Rémi, Palanque, Philippe, Le, Duc-Hoa, and Navarre, David. "PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems." *ICATPN'99*, Williamsburg, VA, USA, June 21-25, 1999. (1999)
23. Valk, Rüdiger. "Petri Nets As Token Objects: an Introduction to Elementary Object Nets." *ICATPN'98*, Lissabon, Portugal, June 1998. Springer (1998)
24. van der Aalst, W. M. P., and Basten, T. "Life-Cycle Inheritance, a Petri-Net Based Approach." *ICATPN'97*, LNCS no. 1248. Springer (1997) 62-81.

Symmetric Communication between Coloured Petri Net Simulations and Java-Processes

Olaf Kummer, Daniel Moldt, and Frank Wienberg

University of Hamburg, Dept. for Informatics, Vogt-Kölln-Str. 30, D-22527 Hamburg,
{kummer,moldt,wienberg}@informatik.uni-hamburg.de

Abstract. In order to widen the applicability of Coloured Petri Nets for the specification and design of large scale distributed applications, a framework has been developed that supports the interaction of Design/CPN and Java processes. The underlying architecture can be used for other tools. Thereby a seamless embedding of the two worlds of Petri nets and object-oriented programming is achieved, allowing problem oriented modelling at different abstraction levels in a fully distributed environment. The general possibilities to connect Coloured Petri Net simulations with remote processes are discussed and a specific implementation of the required framework is sketched. Promising application areas are named and for some of them concrete example models are provided.
Keywords: Coloured Petri Nets, Design/CPN, Distributed Simulation, Framework, Java, Prototyping, Computer Tools, Workflow

1 Introduction

The specification of systems, its evaluation, and its transfer to implementation is still a major task for computer science. One very promising technique in the area of specification, especially when concurrent and distributed systems are involved, are Coloured Petri Nets (see [7]). Because a strong interconnection of specification and implementation is very useful when developing a system, it is desirable to bring together the worlds of Coloured Petri Nets and some popular programming language.

In the area of implementation Java (see [6]) aroused special interest for building applications for the Internet, as it is an object-oriented, reasonably portable programming language that supports additional features such as high-level networking and easy multi-thread programming.

As a specification tool based on Petri nets, we chose Design/CPN (see [3] and [9]), because it is flexible and powerful and comes with a specially adapted graphical editor. Design/CPN supports the development of large systems by means of hierarchical models. Furthermore, an internal programming language, namely ML, can be used to extend the tool for our needs.

In the environment proposed here (for the original version of this paper see [8]), Java should be used for the implementation of graphical user interfaces, database connectivity, and other applications for which one can fall back upon reusable implementations, while Design/CPN serves as the graphical specification

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 86–105, 1999.
© Springer-Verlag Berlin Heidelberg 1999

tool that is powerful in designing and executing models of concurrent, distributed systems.

But there are more reasons why an interaction of Design/CPN and Java has been expected and indeed turned out to be fruitful, namely to overcome some limitations of Design/CPN: The tool is designed for single-user mode only, but especially large-scale projects are dependent on teamwork. Although there is a flexible hierarchy concept, Design/CPN does not really support component re-use, as the exchange of parts between different models is difficult.

In the Design/CPN simulator, interfaces for calling programs implemented in languages other than ML are supported on a very low level only. Although the tool itself, especially with the extension Mimic (see [13]), offers graphical user interface routines, these are also very rudimentary compared to state-of-the-art tools. Last but not least the process of implementing a system for which a Design/CPN prototype has been designed is not well supported by the tool. Since simulation of large-scale models is not quite as efficient as a (compiled) program, it would at least be desirable to provide a stepwise migration of the system from net models to some programming language.

Motivated by all these considerations, a framework has been designed that establishes new possible fields of application where Design/CPN may be employed. The framework extends Design/CPN in several ways: Distributed simulation is achieved by different technical means, namely sockets and pipes. The implemented architecture and alternative concepts for the interconnection of multiple Design/CPN processes are described in section 2. In section 3 it is shown how Design/CPN can communicate with Java processes during simulation. This allows invoking Java methods and even the creation of Java objects. Section 4 explains the reverse communication direction where Design/CPN processes are invoked by Java, viewing the whole Design/CPN process as an object. Possible applications of the extensions and benefits gained from these are discussed in section 5. Section 6 provides an outlook on how the framework can be further extended and how it can be exploited for other approaches.

2 Distributed Design/CPN Processes

If we want to achieve a distributed simulation using Design/CPN, we have to run multiple instances of the program and allow synchronisation of the multiple nets that are being executed.

In our distributed simulation those cases are omitted which include the distributed solving of conflicts. This problem has been treated in e.g. [5]. Instead, a model is split into discrete models and distributed in such a way that only local conflicts can occur, which can be handled within one Design/CPN process.

2.1 Possible Communication Channels

Running multiple instances of Design/CPN in a single simulation run requires communication and synchronisation. Since shared memory is not provided, we have to use some form of message passing.

Whatever I/O-channel is used, a blocking I/O call stops the current net simulation, because the simulator is strictly single-threaded. Hence, it is usually not advisable to perform blocking I/O, although this might be an option in some cases. Instead, we have to do polling I/O, checking for new input occasionally.

There are essentially three possibilities to send and receive messages from within a Design/CPN simulation process:

- Ordinary files. One Design/CPN process might write to a file, while another process might read the file. This is an inefficient method, especially if the processes access a file by NFS. Most network file systems cache file contents so that a change does not immediately propagate through the whole network. This makes ordinary files practically worthless for communication purposes.
- TCP connections. A TCP connection involves a server and a client. The server sets up a server socket under a well-known port number, so that it can accept an arbitrary number of connections from clients. Each server socket is identified by the port number and the server's host at the time the client socket is created.
  Using the ML library `SysIO` Design/CPN supports client sockets with the help of low-level file descriptor I/O.
- Pipes. In Design/CPN it is possible to generate an external process on the same machine by means of the `execute` function. The call will provide input and output pipes from Design/CPN to the created process. These can be handled by the usual ML stream I/O library, which is significantly easier to use than the raw file I/O. Of course, after a few routines have been written to handle the interprocess communication, the designer of a model does not get into contact with these routines very often anyway.
  This approach is suggested in the `process` example that is distributed with Design/CPN 3.02 which starts a Tcl/Tk process as an example.

On the one hand, pipes are most useful when we want to communicate with other processes on the local machine that can be started during the simulation. On the other hand, the more versatile TCP connections have to be used if the processes are required to run on different machines or to be started before the net simulator.

2.2 Possible Communication Architectures

The process started by an `execute` function call could, of course, be another Design/CPN process, but a Design/CPN simulation cannot access its own standard input and output streams. Moreover, at present there is no way to start an interactive simulation automatically within a Design/CPN process with which we could interact. These two difficulties practically rule out direct pipe communication between the two nets.

We cannot use direct TCP communication, because it is not possible to implement server sockets within Design/CPN. This leaves us with three basic options:

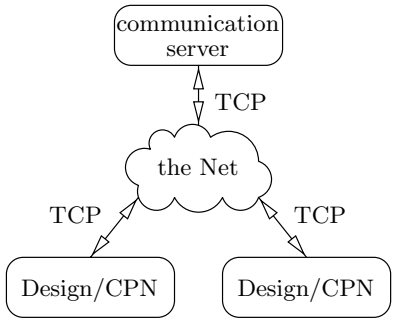


Fig. 1. Server solution

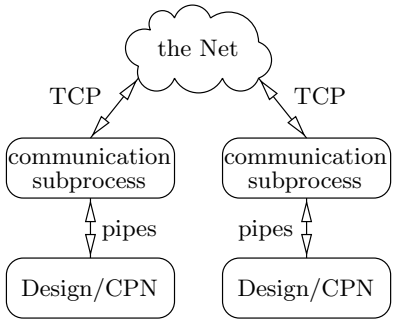


Fig. 3. Fully distributed solution

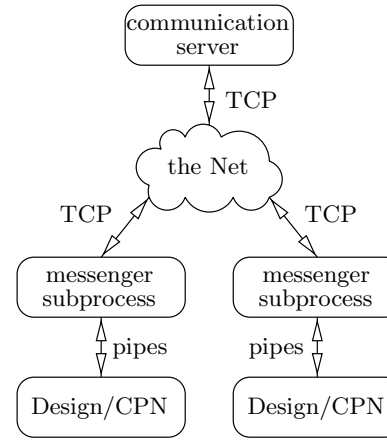


Fig. 2. Messenger solution

- One dedicated communication process is started (the communication server). It opens a server socket that can be accessed by an arbitrary number of Design/CPN processes with one TCP connection each. For a visualisation of this architecture see Fig. 1.
- Every Design/CPN process starts a messenger subprocess and accesses it via pipes. The messenger subprocesses are responsible for transporting the messages to and from the server. It will handle the protocol with the communication server as mentioned above (see Fig. 2).
- Only the subprocesses are present and implement a suitable algorithm for direct message exchange, typically using TCP (see Fig. 3).

In principle, it would be possible to organise the processes like in Figs. 2 or 3 using TCP connections instead of pipes. However, this would imply programming more difficult ML functions without any obvious benefits.

Of course, the communication processes can be implemented in any language that supports the necessary networking ability, but among the many options the most promising seems to be Java for the reasons given in section 1.

The Server Solution. In Fig. 1 we can imagine the Design/CPN processes as actors that exchange messages via a mailbox (the server). Each actor has a

unique mail address, which may be generated by the system at runtime and which will usually exploit the unique network name of the local computer. Some actors may also be assigned a unique well-known address chosen at programming time.

In its simplest form, the mailbox has only to provide commands to send a message to a mail address and to receive a message, if one is available. For performance reasons it may be useful to receive all pending messages. More complex implementations would allow a blocking wait for messages, but this is of little use for a Design/CPN process, since it would completely suspend the simulation. Other options include a forwarding mechanism or a test if a mail address is valid.

The host and the port where the mailbox is located must be public for all Design/CPN processes, so that they can establish a connection.

The mailbox can quickly become the system's bottleneck, especially because polling access is required by Design/CPN. Nevertheless, this method is surprisingly practical and will perform well, provided there are only few processes.

The Messenger Solution. In this case there is still a mailbox, but every Design/CPN process starts a special messenger process by means of the `execute` function, as in Fig. 2. The messenger process can locally communicate with Design/CPN, which is usually much faster than remote interaction. Also, since the messenger is only responsible for a single Design/CPN process, we can use a significantly simpler protocol. Moreover, the messenger can communicate with the mailbox using efficient blocking I/O, thereby taking some load off the bottleneck. Of course, the price to pay is a further indirection leading to some communication overhead.

The Fully Distributed Solution. Here we no longer have a central instance; instead each subprocess spawned by Design/CPN interacts directly with other subprocesses, as shown in Fig 3. This would lead to further performance benefits at the price of a vastly increased complexity of the message-handling algorithm.

Even in a fully distributed architecture there might be a kind of centralised name service, which would allow the distributed nets to communicate without knowing the actual location of the other nets.

2.3 A Communication Package

We have tried the server solution (in C) and the messenger solution (in Java). Here we are going to limit ourselves to a description of the messenger solution, because it results in a more readable Design/CPN package.

The actual implementation of the mailbox and the messenger processes are beyond the scope of this paper. Let it suffice to say that much effort went into the handling of concurrency and into the protocol. The actual network programming turned out to be very easy in Java, contrary to C where network calls constitute the majority of the code.

At the moment the pure subprocess solution is not implemented, but it could be done in about three weeks, if this is required due to performance reasons. The Design/CPN library, which we are now going to describe, and the protocol on

the pipes would not change at all, making an upgrade possible without any noticeable difference for the users.

All basic message handling is done on a single I/O page that can be reused in every distributed system of nets, see Fig. 4. It encapsulates all the transitions to set up and terminate the connection and send or receive messages. As in the following figures, we denote test arcs[1] by arcs without arrow tips.

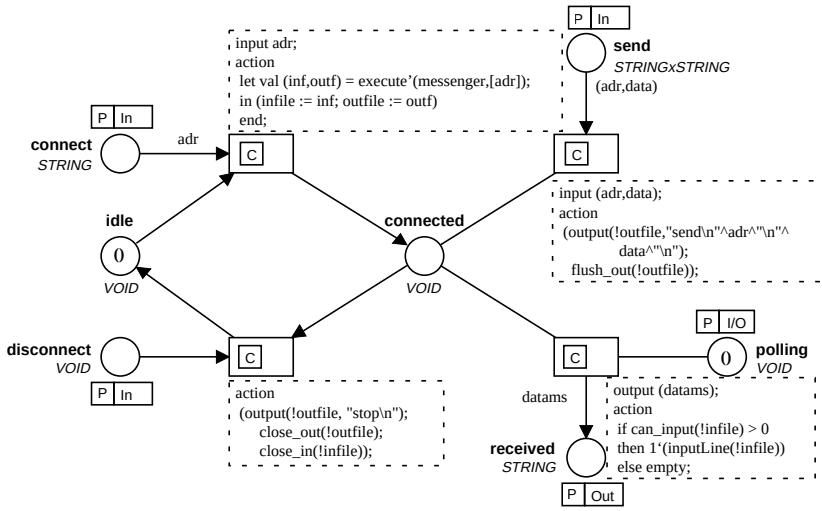


Fig. 4. A subpage for string message I/O (`IOPage`)

In the communication package described here, every Design/CPN process is assigned an arbitrary character string that can be used as the mailbox address. The I/O page is given the mailbox address on which it will listen via the "in" port place `connect`.

The ML variable `messenger` has to contain the path and name of the executable that starts a messenger process and should be defined in the global declaration node, e.g.

```
val messenger = "/home/cpnuser/bin/messenger";
```

This path name is not passed as a token, because it does not often change and because an incorrect path might result in errors undetectable in the ML code.

Outgoing messages must be put into the "in" port place `send` in the form of an address/data pair. Both will be handed on to the messenger through the output stream, preceded by the command `send`. Messages are received from the

[1] In fact, Design/CPN does not support test arcs that do not move any tokens, but an arc without arrow tips is treated as an arc with arrow tips at both ends, thus removing and putting back identical tokens. However, this is not semantically different to a test arc as long as an interleaving semantics is assumed.

messenger through the input stream. It is important to test whether there is any pending input, because the ML function `inputLine` suspends until a newline character (`\n`) is read. The page provides access to the received messages using the "out" port place `received`. The data to be sent and received is always a character string. Other datatypes must be converted to a string format, which can easily be done using the predefined functions provided by Design/CPN.

Tokens can be put into the `polling` place or be removed from there in order to either start or stop polling for new messages. Design/CPN only allows to stop an "automatic" simulation run after a given number of steps or in case there are no more enabled transitions. Removing the `polling` token offers the possibility to stop an automatic simulation run without disconnecting. Furthermore, this feature should be used with nets where it is known when to expect incoming messages, e.g. only as answers to query messages that have been sent before. It is much more efficient not to poll for new messages while the net is working locally and not expecting any message input. If the `polling` port place is not assigned on the "parent" page, the initial marking of one token will be used and polling will always be activated as long as the connection is up. It should be recalled that Design/CPN uses the initial marking of a port place only if no socket place is assigned to it.

Finally, there is a facility to close the connection and shut down the messenger process, because these processes might stay alive when they are not terminated properly. If one wants to do so, one has to change the marking of the `disconnect` place to one token and fire the `disconnect` transition.

Because pipes cannot be uniquely represented as strings, they cannot be used as token colours, hence they have to be stored in reference variables. These are not allowed to be used in arc inscriptions, but in code regions only. We defined two *global* instance variables, `infile` and `outfile`. If one wants to use the subnet more than once, because there is some need to run multiple messenger processes, one can also use *instance* reference variables which have different values for each page instance. This may be desired when many net fragments are developed and tested within one simulator, before they are finally split into many independent nets. The reason why we used global reference variables is that we normally do not use more than one instance of the I/O page and also that instance variables cannot be reached from ML code via Design/CPN's *ML evaluate* feature, e.g. to close the streams manually.

3 Accessing Java from Design/CPN

It should be clear that the message passing scheme used in our architecture does not rely on specific Petri net techniques, so that Design/CPN could be complemented by programs in arbitrary languages that support TCP. Again we choose Java as our example language, even though experiments have also been done with C (for details see [1]).

A straightforward implementation would provide only the basic routines to send and to receive string messages, leaving the programmer with the task of

making the necessary calls. But we can increase the developers' productivity by defining a standard message format and supplying reusable parsing algorithms.

On top of that, we can provide algorithms that actually perform the call that was requested by Design/CPN, so that the message passing framework becomes completely invisible to Java programs. Quite the same can be achieved on the side of Design/CPN where we make a Java method call look like an ordinary substitution transition (see section 3.3).

3.1 The Message Format

The message format should contain all the necessary information to make a Java method call: the object whose method is invoked, the caller that awaits a return message, the method name for the called Java object, and a list of parameters. We have to distinguish these *call messages* from another type of messages which we call *return messages*. A return message is much simpler, as the method name and the caller can be omitted. What remains is just the target object (which should be equal to the original sender of the call message) and a list of parameters, which in the case of Java may only be of length one or zero (if a method is of return type `void`).

One suggestion for a suitable message format that has been implemented has the main aim to keep the net inscriptions and functions simple on the side of Design/CPN. As Design/CPN is based on the functional language ML, it can handle lists very well. Thus, a message is implemented as a list of the components mentioned above.

However, Design/CPN-colours are strongly typed, so a union type of all different types that may appear in a message has to be defined. We end up with a message colour definition as follows:

```
color OBJECT = union Null
                   + RC:RCLASS
                   + RI:RINSTANCE
                   + M:METHOD
                   + Int:INTEGER + Str:STRING + Bool:BOOLEAN + Real:REAL;
color MSG = list OBJECT;
```

with `RC` being a colour that represents a reference to a remote class, `RI` representing a reference to a remote instance, `M` declaring a method name and `Int`, `Str`, `Bool` and `Real` being the constructors for the basic datatypes available in ML and Java. Other types, especially arrays, could be added, if desired.

This very general message format needs additional constraints for well-formed call and return messages, but offers the advantage to define both with the same Design/CPN colour. In order to distinguish call and return messages easily, we chose to put the method name in the first position of the object list instead of using the order of the object-oriented dot notation, where the receiving object is named first. A message starting with a method name is assumed to be a call message, otherwise a return message. The complete sequence of a call message is

`[M(`*`method-name`*`),` *`invoked-object`*`,` *`caller-object`*`,` *`param1`*`,` *`param2`*`, ... ]`

while a return message is simply

```
[receiver-object, result]
```

where *result* is optional and the *receiver-object* is the former *caller-object*.

References are defined as tuples of a class and a process identifier, the latter of which is used to locate remote objects. Unlike a class reference, an instance reference also contains an ID as a third component that makes the triple a globally unique identifier.

```
color RCLASS = product CLASS * PID;
color RINSTANCE = product CLASS * PID * RID;
```

These tuples could have been coded into a single string, but using tuples we can apply the built-in Design/CPN pattern matching capabilities to select the information that is needed to send and receive messages.

All colours that have not been declared are simply defined as `STRING` in our implementation.

Alternatively, the structure of a message given at the beginning of this section could be directly translated into Design/CPN-colours. For a call message, this would result in a four-tuple of the object being invoked, the method name, the calling object and a list of parameters, which again is a list of objects defined as the message above.

Obviously, one call and one return message have to be sent for each statement. When sending messages, we have to provide a unique name for the sender. We can for example use the constant class name `cpn`, construct the "mailbox" name as *hostname*:*process-ID* (of the simulation process), and use a sequence number for every sender being active concurrently. Thus, we end up with a sender like: `RI(("cpn","cpnhost:12345","1"))`. This message has to be handled by a Java process which is waiting for the messages and performs appropriate actions.

All in all, the perfect message format is partly a matter of taste, but we believe that the list format offers most advantages for Design/CPN.

3.2 A Message Translation Page

On top of the message I/O page, another page is built that translates the message to and from the string representation which is used externally (see Fig. 5). Pages like this one must be created individually according to the input and output types that result from the conversion. The pages are expected to share a common structure, so they can be copied from a standard template and then be modified. Since most of the conversion is done by predefined ML functions already available within Design/CPN, the effort is usually negligible.

In our solution, a unique name for connecting to the mailbox is generated by a custom ML function `getUniqueName()` according to the convention described in the previous section. The `disconnect` socket is transformed to a global fusion place, so that the network connection may be terminated from any page in the diagram. Access to the `send` and the `receive` place are provided via global fusion sets, too. Alternatively, the translation page could have defined port places, so

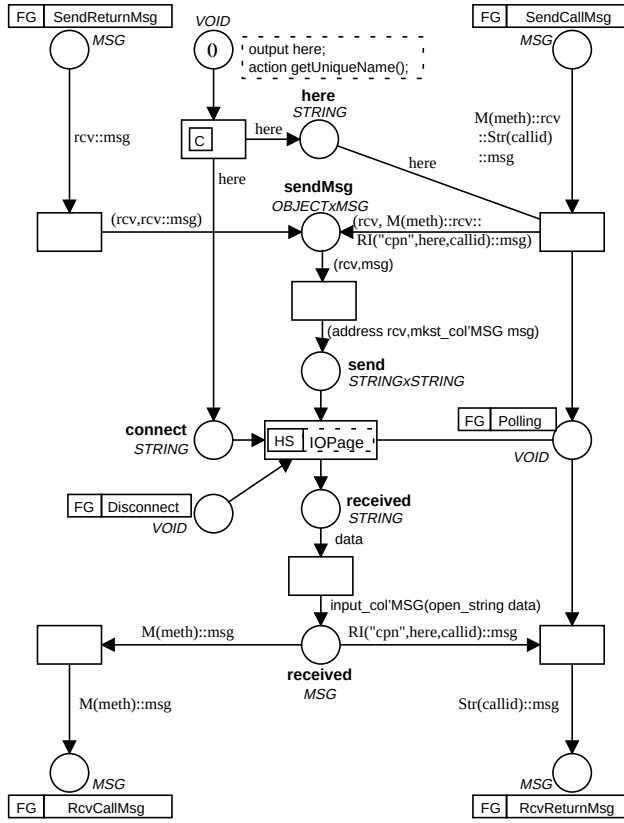


Fig. 5. A message translation page

that it could have been used as a subpage to a net that wants to send messages. We chose the former solution because the translation page is expected to be used by several other pages or page instances and it is neither necessary nor desirable to get multiple instances of this page. A single, global page is of course more efficient in simulation runs.

3.3 A Graphical Petri Net Notation for Method Calls

A further possibility to avoid complex arc inscriptions and to abstract from the message format is to choose a notation that is more adequate for Petri nets. A (Java) method call always goes through the same steps: The message is constructed from the components mentioned in section 3.1, with a unique identifier being constructed as the sender. Then, the message is sent and the caller waits for a return message, which is again decomposed into its components.

Fig. 6 shows a subpage that implements this behaviour. The interface consists of all components of a call and the corresponding return message. The maximum

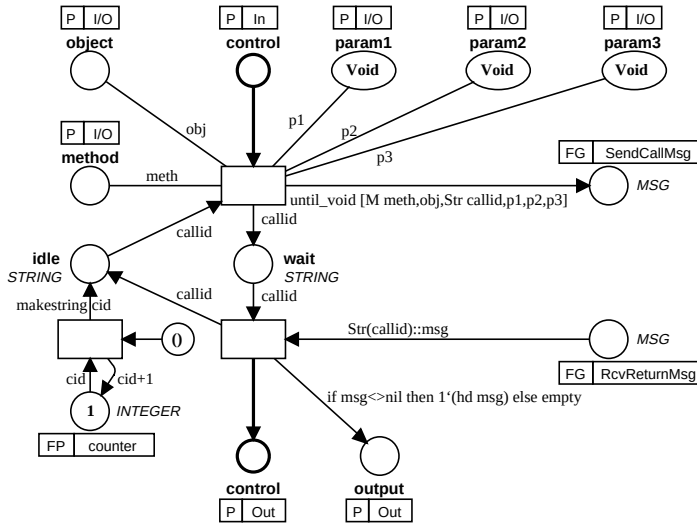


Fig. 6. A subpage wrapping calls to Java methods (`Call`)

number of parameters has been chosen to be three in this net but may easily be extended. No practical problems are to be expected, since the maximum of parameters needed can be determined at compile-time and should not be too high, anyway. Moreover, note the distinction between data and control flow: A special place named `control` indicates the control state of the call, while all other input data is read by test arcs only. The control places' colour, `VOID`, is defined to contain one element only, which is referred to as the *black token*.

A unique ID is assigned to each instance of the page by using the page instance fusion place `counter`. Each page instance may only invoke one method at a given time (asserted by the place `idle`), but several instances of the `Call` subpage may act concurrently. To handle parameter input, we again take advantage of the Design/CPN feature that the initial marking of a port place is only used if the port place remains unassigned. Thus, all parameter places that are not used by a substitution transition remain `Void` and are suppressed in the construction of the call message by the ML function `until_void` defined in the global declaration node. The `output` port place should only be used if the called method actually returns a result, because then the arc expression will produce no token (not even a black one). A separate control token is produced to indicate that the method call is finished. It may be used to enable the next transition.

To make the nets calling Java more readable, some Design/CPN regions have been suppressed in the following figures. However, a special graphical notation makes the syntax of the nets clear without ambiguity, as may be seen in Fig. 7. The left-hand side contains all regions and inscriptions, while the right-hand side shows how these are translated into graphical representations: All control places and control arcs are shown in bold style. They are always of colour `VOID`, as

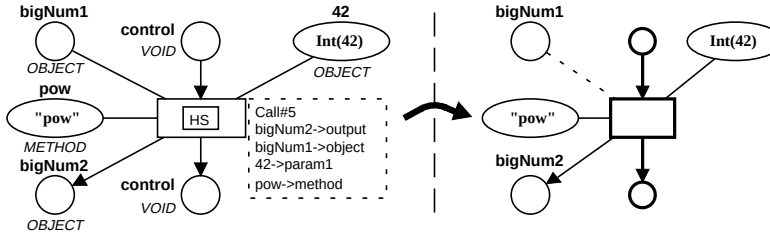


Fig. 7. An example of the special graphical notation for using the `Call` subpage

they store control information only. Data places and arcs are shown in normal style and all have the colour `OBJECT` unless they are a `method` place, which can be recognised by their position directly to the left or to the right to a `Call` (see Fig. 6) substitution transition. If an arc has no arc inscription, it is either connected to a substitution transition (Design/CPN neglects inscription of such arcs, anyway) or it has the hidden arc inscription `()` which is one black token.

Any transition that invokes an object method, further to referred to as a method invocation transition, is indicated by the presence of a dotted arc without arrow tips. Since all these transitions are substitution transitions refined by the `Call` subpage, the hierarchy substitution region (`HS`) may be omitted. In order to still state a precise port assignment, the following rules apply: The arc connecting a socket place to the `object` I/O port is exactly the dotted one. The input and output control sockets can be recognised as the bold places with input and output arcs, respectively. The input parameters are connected with test arcs in normal style. If more than one input parameter is used, the arcs have to be labelled with the parameter index `p`n (this notation is not needed in this paper). The output port is assigned to the socket that is connected to the substitution transition via an output arc in normal style.

3.4 An Example Net Calling Java

In the example net in Fig. 8, a computation with Java's big integers is performed, but this time, concurrency is exploited. To specify concurrency, we use transitions to fork and to join control flow. Since both "threads" send messages to Java, we have to take care that the return messages are correctly associated. This is ensured by every instance of the `Call` subpage using a different identifier.

Of course, from the viewpoint of efficiency, Java method calls on a small scale as in this example do not really make sense, since the overhead of sending and interpreting messages is very high. Since efficiency was clearly not among the goals of our prototype implementation, we did not analyse performance issues.

There have been other test applications which better exploit the advantages of our framework, but are too extensive to be presented here. In [4] a game was developed that simulates a stock exchange using a Coloured Petri Net. The net was augmented by calls to a graphical user interface programmed in Java, resulting in a game that can be played by multiple players across a network. The

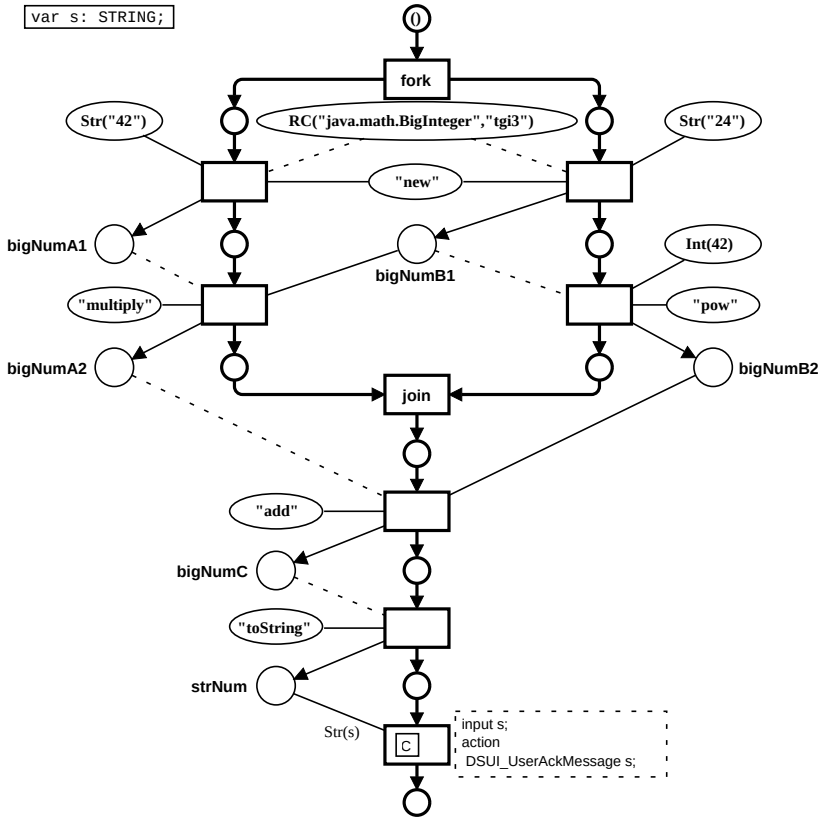


Fig. 8. A Petri net performing concurrent calculations with Java `BigIntegers`

new net shows a dramatic increase in usability that would have been impossible or at least cumbersome with pure Design/CPN. Using Java only, the development time might have been shorter, but the CPN model offers a much better understanding of concurrency and distribution in the resulting running system.

3.5 Processing Method Calls in Java

We now briefly describe how the Java side of our framework treats the incoming messages. As a first step, a message is converted into an internal representation using a straightforward top-down-parser.

Now some remote references might point to an object of the local Java process. Class references are immediately resolved to ordinary Java classes.[2] Then the framework has to translate the remote instance references into local Java objects using a special table of externally known objects.

[2] Java allows the loading of classes at runtime requiring only a string representation of the class name. Even the creation of classes at runtime is possible.

Afterwards the framework determines the class of the invoked object and uses the standard `reflection`[3] package to get a list of all public methods which the object supports. It chooses the appropriate method from the list and calls it in an individual thread. Now other calls can be concurrently received and parsed.

After the completion of the method call, the result is sent back to the caller whose address was specified in the message. If a reference to a Java object is returned, the framework generates a unique external name for it and inserts this name in the table of externally known objects. Design/CPN will then receive the message and forward the result.

If a Design/CPN process interacts with multiple Java processes, remote references might point to an object of a remote Java process instead of a local Java object. This case, too, is handled by our framework, e.g. one could, if desired, store a reference to an object on one process in a hash-table on another process.

There are two possibilities for a CPN process to obtain references to Java objects. To start with, such an object may be created by a special program that also invokes the framework and inserts the object into the table of externally known objects under a well-known name. Then, the CPN process may access the `new` method of a class reference. In fact, the message looks like a normal method call, which is not the case in the Java programming language, where `new` is a keyword with a special syntax. To keep the message format clean, we decided to use `new` like a static method provided by every class instead.

This allows the creation of arbitrary Java objects from a Design/CPN net, even for built-in classes like hash-tables, windows, etc. Although this device is extremely powerful, a note of caution has to be: The creation of new objects using the `new` method provides complete access to the Java environment, thereby opening up huge security holes. But Java programs can deliberately reduce their access rights by means of the `SecurityManager` interface. If this does not prove sufficient, it is still possible to protect the access to either the Java framework or to the entire message handling system by passwords or other techniques.

4 Accessing Design/CPN from Java

In the last section, we have shown how to call Java from Design/CPN. This approach is useful when the major part of the application is modelled with Design/CPN and Java is used to complement the application.

In other cases, an application or at least large parts may have already been implemented in Java, but Petri nets are utilised for modelling use cases or workflows that are contained in the application. Then, one may wish to design and run these parts with Design/CPN and call Petri nets from Java. The basic idea is that a system modeller can use the most appropriate tool for representing and solving the problem at hand.

[3] The `reflection` mechanism is a powerful feature of the Java environment that allows the complete analysis of objects, classes and methods at runtime. Thus, Java programs can view a *reflected* image of themselves as if in a mirror. Additionally, the modification, the creation, and the invocation of objects is supported.

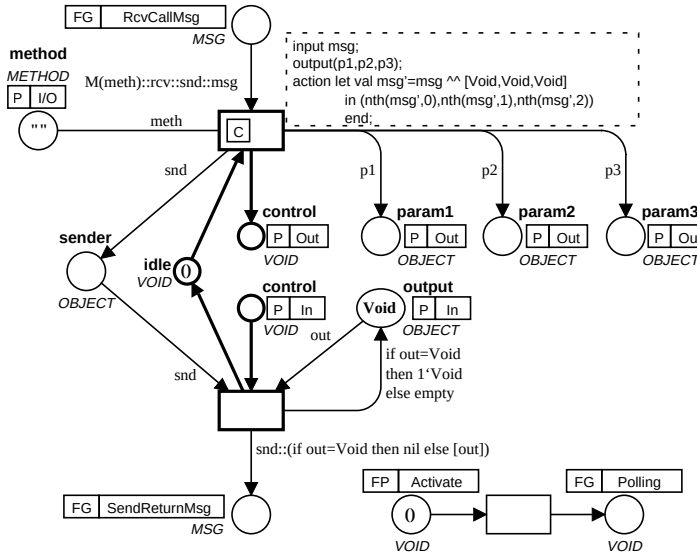


Fig. 9. A subpage for calls to a CPN (`CPNCall`)

All one has to do to extend our approach to support this feature, is firstly to implement generic Java proxy objects that call Design/CPN nets via messages. Secondly, Design/CPN nets have to be extended to be able to receive and process call messages, not only return messages. In a sense, a Design/CPN process has to behave like a single (static) object, thus providing some methods that may be called.

4.1 Designing a Subpage for Calls to Design/CPN Nets

Again, we have tried to find a very general solution, i.e. a net subpage that can be re-used for any Design/CPN net implementing some method call. Fig. 9 shows such a subpage which is basically the counterpart to the net presented in Fig. 6. The transition at the bottom ensures that the net is able to receive messages all the time, in contrast to a net that just calls Java methods and polls for return messages if some call has been sent. The upper transition checks all incoming messages whether they are call messages to the method name given in the port place `method`. If so, the parameters contained in the call message are distributed into separate places (again, the maximum number of parameters is restricted to three) and a token is put into the `control` output port place to specify that the "parent" net may start now. The receiver of the message is ignored, because the whole net is treated like a single object. If this net had not been the receiver of the message, the message would not have been delivered there. The `idle` place is needed because this simple version of calling a net does not support concurrent calls to the same net, while it does allow concurrency within the net. This restriction is lifted when our object Petri net approach is

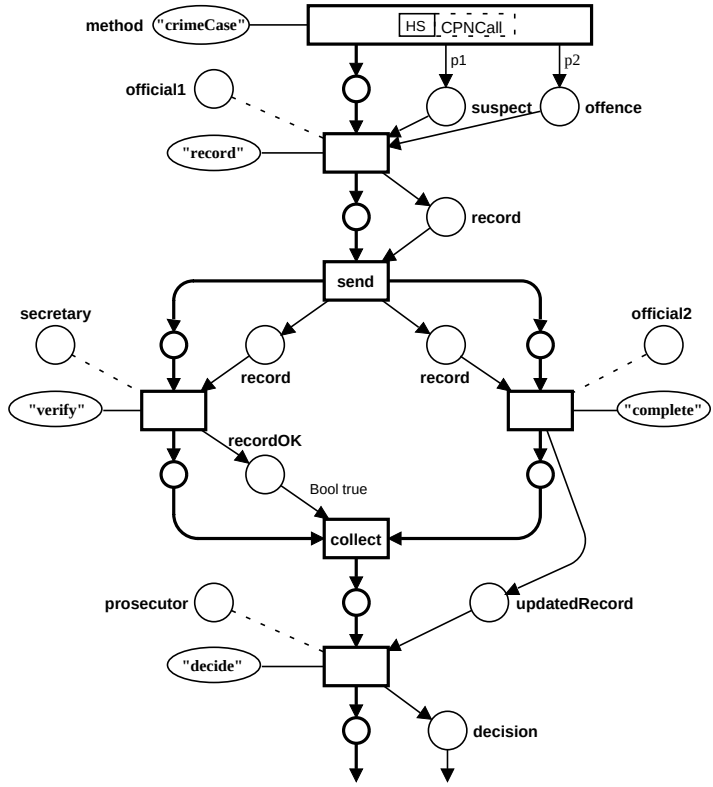


Fig. 10. An example of a net that can be called from Java

used as discussed in the conclusion. After the parent net has finished its task, it has to put a token into the `control` input port place. It should remove all input parameter tokens as well as all tokens that were produced during the run, lest the next call fails or produces unexpected results.[4] When a socket place is assigned to the `output` place, some token has to be present that is used as a return parameter. The subpage then constructs a return message to the sender of the call message and puts it into the global fusion place `SendReturnMsg`, so that it is sent by the `Message` page and the `IOPage`.

4.2 An Example of a Net that Can Be Called

To illustrate the use of this subpage, we specify a net that implements a method executing some example workflow by W. v.d. Aalst, cited in [15].

The example was introduced as follows. When a criminal offence happens and the police has a suspect, a record is made by an official. This is printed and sent to the secretary of the Justice Department. Extra information about the

[4] It is not too complicated to check this property automatically using place invariants.

history of the suspect and some data from the local government are supplied and completed by a second official. Meanwhile the information on the official record is verified by a secretary. When these activities are completed, a prosecutor determines whether the suspect is summoned or charged, or whether the case is suspended.

In Fig. 10, a workflow for this case is modelled using our object-oriented Petri net notation introduced in section 3.3, extended by the feature that a method invocation transition may consume input parameters. In the upper part, the interface places of the method `crimeCase` are connected to the subpage `CPNCall` through a substitution transition. The arcs that point to the bottom border of the figure are actually connected to this transition, too. Note that tokens for the places `official1`, `official2`, `secretary`, and `prosecutor` have to be provided elsewhere. We do not give any further details on how the method invocation transitions might be refined. In fact, the activities may be implemented as code regions, as subpages, in Java or even as other nets that may be called through the message interface.

In this case the call of the workflow from Java simply looks like

```
RemoteCPN cpn=new RemoteCPN("cpnhost:12345");
String decision=(String)cpn.execute("crimeCase","Roger Rabbit",
                                    "murder of Marvin Acme");
```

assuming that `cpnhost:12345` is the hostname / process-ID of the simulation process (and thus its mailbox name) simulating the workflow net, that the suspect is Roger Rabbit, and that he is accused of murder, where the decision of the prosecutor (as well as the input parameters) is implemented as a `String`.

5 Possible Applications and Benefits

Petri nets are already a concurrent formalism, so a net model documents the intended concurrency without the need to distribute it. But there remain at least two cases where a distributed net model is favourable even in the absence of interaction with Java:

- Need for performance gains. Whether any significant performance improvements are possible depends on several factors. It must be possible to split the application into parts that communicate by messages. This is often naturally the case, but existing net models might not show the possible splitting lines clearly. What is more important is that the nets should require as little communication messages as possible. Alternatively, the net might be demanding in terms of ML evaluation time, e.g. animation or optimisation algorithms. In both cases, the communication cost might be dominated by the computation cost.
- Real concurrency. The Design/CPN simulator is a single-threaded application. Thus, no real concurrency, not even multi-threading is available. Multi-threading would be especially desirable if complex or time-consuming ML functions are executed, which is normally done within one simulation step,

delaying all transitions. Now, code may be moved to a separate diagram (or any external program, see below), enabling the calling net to continue working while the computation is performed. The drawback of polling for the answer is in some cases preferable compared to blocking the whole simulation.
- Necessary distribution. Some applications require access to distributed resources, e.g. a visualisation module that accesses various screens or a game that needs input from many players. Such applications cannot naturally be realised without a distributed simulator.

However, the greatest benefit of our communication framework comes from the interoperability with other programming environments:

- The access to Java processes enables the developer of a net model to incorporate much more complex GUIs into a net. This improves the interaction with the user of the net, but might also be used to animate and visualise the simulated process or to generate more elaborate statistics and debugging information.
  Although there are some GUI libraries for Design/CPN already—the most notable one being Mimic/CPN described in [13]—they do not match the flexibility of Java or comparable languages. Moreover, there is a lack of rapid prototyping tools which greatly speed up the GUI development.
- Distributed computing also allows multiple users to participate in a single simulation from different terminals.
- Processes that are controlled by Design/CPN might also handle general I/O devices. This may simplify the control of a system by a net, a possibility already mentioned in the context of the security system presented in section 1.5 of volume 3 of [7]. There it was proposed to extract parts of the ML code generated from the net for the execution on a stand-alone microprocessor. Such a task might be considerably simplified if the connection to the outside world remains the same during the translation.
- It becomes possible to reuse code that was not developed in Design/CPN or ML. In this area the standard container classes come to mind immediately, but in fact there is a wide range of programs for Java covering almost all aspects of algorithms and data structures as well as various I/O and network libraries.

There are additional benefits when Coloured Petri Nets can also be called from the outside:

- It is possible to move gradually from a Petri net prototype to an implementation using Java. Although this will require that the nets themselves are structured in an object-oriented way, it remains a viable route.
- The specification of workflows with Petri nets has attracted much interest in the past years. Using the framework it seems possible to use Coloured Petri Nets for executable prototypes of workflows and in the future maybe even for final products.

6 Outlook

The framework presented here improves the usability of Design/CPN in some of the most important areas: distribution, interoperability, appropriate modelling, and graphical user interfaces. A smooth transition from a specification within Design/CPN via distributed Design/CPN modules to distributed Java modules seems possible. Some applications have already been implemented thereby documenting the gained flexibility.

In the meantime, we developed a high-level Petri net editor and simulator, Renew [14], that offers an even closer connection to Java and object-orientation in general. Java is used as an inscription language, so that Java objects can be accessed directly, but distribution is not supported directly. Renew supports dynamic net instances and net references, which eases introducing object-oriented concepts into Petri nets. In the near future we are going to use the framework presented here to connect Design/CPN and Renew. In order to achieve object-oriented behaviour of nets, we are going to extend the framework to object-oriented Petri nets in the sense of [10] and [11]. In those approaches the structure of the nets represents most object-oriented features without extensions of Coloured Petri Nets themselves. Tool support of the object-oriented techniques could then further simplify the development process.

Additionally, Artificial Intelligence concepts as already used in [12] are going to be extended by providing a connection to Prolog. This will allow us to access available tools for logic programming directly from Petri net models.

If Design/CPN itself was extended by multi-threaded simulation of net models, this would capture some concurrency aspects of our framework. The distribution and interaction with Java would still be an essential benefit of our framework, which would even be improved, because blocking input could be used instead of polling.

Our communication framework already bears some resemblance to other distributed object-oriented architectures. Although many functions and just as many concepts are still missing, it does no longer seem far fetched that Petri nets might one day be usable with systems like CORBA [2].

Acknowledgments

Much of the Java implementation described in this paper was done during a project at the University of Hamburg. We would like to thank the participants for their efforts, especially Matthias Ernst, Torben Kroll, Heiko Rölke, and Eberhard Wolff.

A message passing library in C was implemented by Heinrich Biallas and has been documented in [1].

In [4] Margret Freund-Breuer and Olaf Fricke developed the stock exchange model that was used as a first test case for our Java interaction framework.

References

1. Heinrich Biallas. Realisierung der verteilten Ausführung von gefärbten Petrinetzen. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, November 1997.
2. The OMG Corba Page. WWW page at `http://www.omg.org/corba/`. Contains references to the current specification of CORBA/IIOP (on 05-15-98, this is version 2.2), a discussion of the Object Management Architecture, and many more links.
3. Design/CPN Online. WWW page at `http://www.daimi.au.dk/designCPN/`.
4. Margret Freund-Breuer and Olaf Fricke. Spezifikation mit gefärbten Petri-Netzen am Beispiel des Börsenspiels. Studienarbeit, Fachbereich Informatik, Universität Hamburg, September 1993.
5. Dirk Hauschildt. A Petri Net Implementation. Fachbereichsmitteilung FBI-HH-M-145/87, Universität Hamburg, Fachbereich Informatik, 1987.
6. The Java Home Page. WWW page at `http://java.sun.com`. Contains references to online material as well as to many introductory books and technical papers.
7. Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992: Vol. 1, 1994: Vol. 2, 1997: Vol. 3.
8. Olaf Kummer, Daniel Moldt, and Frank Wienberg. A Framework for Interacting Design/CPN- and Java-Processes. In Kurt Jensen, editor, *Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, pages 131–150, Department of Comupter Science, University of Aarhus, 1998. number 532 in DAIMI PB.
9. Meta Software Corporation, Cambridge, MA, USA. *Design/CPN Handbook Version 2.0*, 1993.
10. Daniel Moldt. *Höhere Petrinetze als Grundlage für Systemspezifikationen*. Dissertation, Universität Hamburg, Fachbereich Informatik, August 1996.
11. Daniel Moldt and Christoph Maier. Coloured Object Petri Nets - A Formal Technique for Object Oriented Modelling. In B. Farwer, D. Moldt, and M.-O. Stehr, editors, *Petri Nets in System Engineering, Modelling, Verification and Validation*, pages 11–19, Fachbereich Informatik, Univ. Hamburg, 1997. FBI-HH-B-205/97.
12. Daniel Moldt and Frank Wienberg. Multi-Agent-Systems based on Coloured Petri Nets. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, number 1248 in Lecture Notes in Computer Science, pages 82–101, Berlin, 1997. Springer Verlag.
13. Jens Linneberg Rasmussen and Mejar Singh. *Mimic/CPN – A Graphics Animation Utility for Design/CPN. User's Manual Version 1.5*. Computer Science Department, Aarhus University, December 1995.
14. The Reference Net Workshop. WWW page at `http://www.renew.de`. Contains tool documentation including an introdution to the refernce net formalism.
15. Rüdiger Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In Jörg Desel, editor, *19th International Conference on Application and Theory of Petri nets*, number 1420 in LNCS, Berlin, 1998. Springer-Verlag.

Recent Developments in Modeling and Analysis of Hybrid Dynamic Systems

Bruce H. Krogh

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-8390
krogh@ece.cmu.edu

Abstract. Recent research in hybrid dynamic systems has brought together formalisms and techniques from computer science and control theory to address problems involving a mixture of discrete and continuous state variables. Computer scientists have extended standard models of finite-state systems to include continuous dynamics that determine when discrete state transitions can occur. Control theorists have introduced switching logic and discrete states to select continuous dynamic modes in models of controllers and physical systems. The interaction of discrete and continuous phenomena–and the interaction of computer scientists and control theorists–have led to new research problems and new research results.

Three models illustrate different perspectives in the hybrid systems literature: block diagrams with discrete and continuous dynamic blocks; hybrid automata with continuous dynamics associated with each discrete state; and continuous Petri nets with continuous dynamics associated with each place. As illustrated by selected examples from the literature, each formalism offers intuitive features for modeling particular classes of hybrid dynamic systems.

Computational tools have been developed to model, simulate, and analyze various classes of hybrid systems. To analyze hybrid systems, control theorists have introduced extensions to Lyapunov theory for stability analysis, and computer scientists have extended formal verification techniques to certain classes of hybrid systems. In the latter research, decidability results clearly distinguish tractable from intractable problems.

This talk reviews the models, analytical results, and computational tools that have been developed recently for hybrid dynamic systems. The state of the research and prospects for future advances in the theory and applications will be assessed.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 106–106, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Autonomous Continuous P/T Systems

Laura Recalde, Enrique Teruel, and Manuel Silva*

Dep. Informática e Ingeniería de Sistemas
Centro Politécnico Superior de Ingenieros, Universidad de Zaragoza
María de Luna 3, E-50015 Zaragoza, Spain

Abstract. Discrete event dynamic systems may have extremely large state spaces. For their analysis, it is usual to relax the description by removing the integrality constraints. Applying this idea, continuous P/T systems are defined by allowing fractional firings of transitions, and thus the existence of non-discrete markings [4,5,1]. In this paper we compare the behaviors of discrete and continuous systems, and observe that they are not necessarily similar. The problems that appear lead to the definition of two extensions of reachability. Many properties shall be extended differently depending on which reachability definition is being considered. Here, we concentrate on liveness and deadlock-freeness, proposing extensions and relating them to their discrete counterparts.

1 Introduction

One of the most important tools for the analysis of P/T systems is the state equation, which is based on the relaxation of the reachability condition using a *path integration* approach. This description is sometimes further relaxed by dropping integrality constraints, following the approach that is typical in the mathematical modeling of systems with large state spaces (e.g., population models). This *fluidization* allows to use *linear* programming instead of *integer* programming in the verification of certain properties.

These principles can also be applied in the reverse order, first continuization and then path integration. By disregarding first the integrality of variables, we get *continuous P/T systems* [4,5,1]. In these models, "fluid tokens" are contained in "deposits" (the places), the "level" of which (the marking) captures the state of the system. Transitions are regarded as "mixing valves" whose firing (opening) consumes fluid from the input places and produces fluid onto the output places in a given proportion, defined by the arc weights. These nets are interesting in the modeling of certain continuous systems, and also as an approximation of systems with large amounts of (discrete) tokens.

Autonomous continuous P/T systems were introduced in [4]. Although some work has been done in the analysis of timed continuous P/T systems [4,5,1], almost nothing has been done w.r.t. the analysis of autonomous continuous P/T

* This work was partially supported by the project HF97-169, within the Picasso programme and project CICYT TAP98-0679

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 107–126, 1999.
© Springer-Verlag Berlin Heidelberg 1999

systems. It might be thought that they cannot be that different from discrete P/T systems. However we will see that the behavior of a system considered as discrete may be completely different from its behavior if it is considered as continuous. Moreover, although the extension of the "token game" to continuous P/T systems is quite immediate, it is not so clear how such a basic concept as reachability should be extended. For instance, should be considered reachable a marking that cannot be obtained firing a finite sequence, but is obtained after an infinitely long one?

The basic definitions of autonomous continuous P/T nets and systems are introduced in Section 2. Some immediate properties are also proven, as the convex nature of the reachability space, or the equivalence, in case every transition is fireable, of behavioral and synchronic relations (in particular boundedness and str. boundedness). In Section 3 some examples are presented which show that the properties of a system may be very different depending on whether it is considered as discrete or continuous. A new definition of reachability, *limit reachability*, in which infinitely long firing sequences are allowed, is introduced in Section 4. If every transition is fireable, the limit reachability space of consistent systems is characterized as the set of solutions of the state equation. Section 5 is devoted to the analysis of liveness. Two definitions of liveness are introduced that correspond to the two views of reachability (Subsection 5.1). In Subsection 5.2 it is observed that both kinds of liveness are preserved if the marking is scaled. The relationship that exists among all the definitions of liveness (the two continuous definitions and the discrete one) is analyzed in Subsection 5.3. In Subsection 5.4 two necessary conditions are obtained for the liveness definition that seems more convenient. Finally, we restrict to subclasses, in particular, equal conflict [13], and free choice [6] systems, for which stronger results can be proved (Subsection 5.5).

2 Definition and First Results

A continuous P/T system is a pair $\langle \mathcal{N}, \mathbf{m_0} \rangle$, where $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a P/T net (P and T are disjoint (finite) sets of *places* and *transitions*, and **Pre** and **Post** are $|P| \times |T|$ sized, natural valued, *incidence matrices*), and $\mathbf{m_0}$ is a *continuous marking*.

The net in a continuous P/T system is the usual P/T net. In particular,the restriction on the arc weights being integer is maintained. This is particularly reasonable when the continuous P/T system is used as an approximation of a discrete system. In the case of continuous (or hybrid) P/T systems used to model continuous systems, the integrality of the arc weights is not a big restriction because rational arc weights could be multiplied by the least common multiple of their denominators.

All the concepts based on the representation of the P/T net as a graph (strong connectedness, presets, postsets, ...) can be directly applied to continuous P/T nets. In particular, the definitions based on the annullers of the token-flow matrix ($\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$) can be immediately extended. Right and left natural annullers

are called T- and P-semiflows, respectively. When $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, $\mathbf{y} > \mathbf{0}$ the net is said to be *conservative*, and when $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$, $\mathbf{x} > \mathbf{0}$ the net is said to be *consistent*. The definitions of subclasses that depend only on the structure of the net are also generalized. For instance, in choice free nets (CF) each place has at most one output transition, free choice nets (FC) are ordinary nets in which all conflicts are equal ($^\bullet t \cap {}^\bullet t' \neq \emptyset \Rightarrow \mathbf{Pre}[P,t] = \mathbf{Pre}[P,t']$), and equal conflict nets (EQ) are the weighted counterpart of FC nets.

A continuous marking is a $|P|$ sized, non-negative, *real* valued, vector. A *continuous P/T system* is a pair $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$, where $\mathbf{m_0}$ is the initial continuous marking. A transition t is *enabled* at $\mathbf{m}$ iff for every $p \in {}^\bullet t$, $\mathbf{m}[p] > 0$. In other words, the enabling condition of continuous systems is the same as the enabling condition of discrete ordinary systems: every input place is marked. As in discrete systems, the *enabling degree* of a transition measures the maximal amount in which the transition can be fired in one go, i.e. $\text{enab}(t, \mathbf{m}) = \min_{p \in {}^\bullet t}\{\mathbf{m}[p]/\mathbf{Pre}[p,t]\}$. The firing of t in a certain amount $\alpha \leq \text{enab}(t, \mathbf{m})$ leads to a new marking $\mathbf{m}' = \mathbf{m} + \alpha \cdot \mathbf{C}[P,t]$. This is denoted as $\mathbf{m} \xrightarrow{\alpha t} \mathbf{m}'$. Notice that a transition being enabled or not does not depend on the arc weights, although they are important to compute the enabling degree and to obtain the new marking. A certain marking $\mathbf{m}'$ is *reachable* from $\mathbf{m}$ if a (finite) fireable sequence exists leading from $\mathbf{m}$ to $\mathbf{m}'$.

Definition 1. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous system. A certain marking $\mathbf{m} \in (\mathbb{R}^+ \cup \{0\})^{|P|}$ is* reachable *iff a finite sequence $\sigma = \alpha_1 t_1 \alpha_2 t_2 \ldots \alpha_k t_k$, exists such that*

$$\mathbf{m_0} \xrightarrow{\alpha_1 t_1} \mathbf{m}_1 \xrightarrow{\alpha_2 t_2} \mathbf{m}_2 \cdots \xrightarrow{\alpha_k t_k} \mathbf{m}_k = \mathbf{m}$$

where $t_i \in T$ and $\alpha_i \in \mathbb{R}^+$.

The reachability space*, $\text{RS}_\text{C}(\mathcal{N}, \mathbf{m_0})$, is the set of all the reachable markings.*

Given σ such that $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$, and denoting by $\boldsymbol{\sigma}$ the firing count vector of σ, then $\mathbf{m}' = \mathbf{m} + \mathbf{C} \cdot \boldsymbol{\sigma}$. This is known as the state equation *of $\mathcal{S}$.*

The set of all the markings $\mathbf{m} \in (\mathbb{R}^+ \cup \{0\})^{|P|}$ that fulfil the state equation, with $\boldsymbol{\sigma} \in (\mathbb{R}^+ \cup \{0\})^{|T|}$, is called the linearized reachability space *(w.r.t. the state equation), $\text{LRS}_\text{C}(\mathcal{N}, \mathbf{m_0})$.*

If $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, then $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$. Thus, as in discrete P/T systems, $\text{RS}_\text{C} \subseteq \text{LRS}_\text{C}$.

The possibility of firing the transitions in any amount (up to the enabling degree) leads to the fulfilment of several properties on the set of fireable sequences related to *homothecy* and *monotony*:

Proposition 2. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous P/T system, and σ a sequence fireable at $\mathbf{m_0}$.*

- *For any $\alpha \geq 0$, $\alpha\sigma$ is fireable at $\alpha\mathbf{m_0}$, where $\alpha\sigma$ represents a sequence that is equal to σ except in the amount of each firing, that is multiplied by α.*
- *If $0 \leq \alpha \leq 1$, $\alpha\sigma$ is fireable at $\mathbf{m_0}$.*

– *For any* $\mathbf{m_0}' \geq \mathbf{m_0}$, σ *is fireable at* $\mathbf{m_0}'$.

This endows the reachability space with a particular structure that it does not have in discrete systems: it is a *convex set*. That is, for any two markings that can be reached from $\mathbf{m_0}$, any intermediate marking that can be expressed as their linear combination is reachable too.

Theorem 3. *The reachability space of a continuous P/T system is a convex set.*

Proof. Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous system and $\mathbf{m}_1, \mathbf{m}_2$ two reachable markings, i.e., $\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}_1$ and $\mathbf{m_0} \xrightarrow{\sigma_2} \mathbf{m}_2$. Let $\alpha \in [0,1]$. Then, $\alpha\mathbf{m_0} \xrightarrow{\alpha\sigma_1} \alpha\mathbf{m}_1$ and $(1-\alpha)\mathbf{m_0} \xrightarrow{(1-\alpha)\sigma_2} (1-\alpha)\mathbf{m}_2$. Therefore, $\alpha\mathbf{m}_1 + (1-\alpha)\mathbf{m}_2$ is reachable from $\mathbf{m_0}$ firing $\alpha\sigma_1 + (1-\alpha)\sigma_2$. □

The same idea of firing just a part of what is enabled is the basis of the following algorithm, that checks whether every transition is fireable at least once.

The algorithm fires the enabled transitions, which can lead to the enabling of other transitions, but taking care not to disable any of the former. Thus, the set of enabled transitions, T^j, never decreases. If it does not increase, a point has been reached in which the firing of the enabled transitions cannot lead to the enabling of any other one, therefore not every transition can be fired. Otherwise, since the number of transitions is finite, the algorithm stops when all have been considered.

Algorithm 1

Input: A continuous P/T system, $\langle \mathcal{N}, \mathbf{m_0} \rangle$
Output: The set of dead transitions, T'
Begin
  Let $T^0 = \emptyset$;
  Let $T^1 = \{t \mid \text{enab}(t, \mathbf{m_0}) > 0\}$
  $j := 1$
  While $T^j \neq T$ and $T^j \neq T^{j-1}$ **do**
    Let σ_j be a sequence obtained firing all the transitions in $T^j \setminus T^{j-1}$
      with half their enabling degree, and let $\mathbf{m}_{j-1} \xrightarrow{\sigma_j} \mathbf{m}_j$
    $T^{j+1} = \{t \mid \text{enab}(t, \mathbf{m}_j) > 0\}$
    $j := j + 1$
  od
  $T' := T \setminus T^j$
End

In other words, in continuous systems it is equivalent that every transition is fireable or that a strictly positive marking can be reached. From this, realizability of T-semiflows can be deduced, and all the three are equivalent if the net is consistent.

Proposition 4. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a continuous system.*

1. *It is equivalent that every transition is fireable or that a strictly positive marking can be reached.*
2. *If every transition is fireable, for every* $\mathbf{x} \geq \mathbf{0}$ *such that* $\mathbf{C} \cdot \mathbf{x} \geq \mathbf{0}$ *a marking* $\mathbf{m} \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$ *exists such that* $\mathbf{m} \xrightarrow{\sigma}$ *and* $\boldsymbol{\sigma} = \alpha \mathbf{x}$ *with* $\alpha > 0$*. Moreover, both properties are equivalent if the net is consistent.*

Fireability of T-semiflows, implies that behavioral and structural synchronic relations [11] coincide in continuous systems in which every transition is fireable at least once. In particular, defining boundedness and str. boundedness as in discrete systems (a system is bounded iff $\mathbf{k}$ exists such that for every reachable marking $\mathbf{m} \leq \mathbf{k}$, and it is str. bounded iff it is bounded with every initial marking) it is immediate to see that that both concepts coincide in continuous systems in which every transition is fireable. And, as in discrete systems, str. boundedness is equivalent to the existence of $\mathbf{y} > \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} \leq \mathbf{0}$ (Farkas Lemma [8]).

Theorem 5. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a continuous system in which every transition is fireable at least once. It is equivalent:*

- $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *is bounded.*
- $\mathcal{N}$ *is bounded with any initial marking (str. bounded).*
- $\mathbf{y} > \mathbf{0}$ *exists such that* $\mathbf{y} \cdot \mathbf{C} \leq \mathbf{0}$*.*

3 Discrepancies between Continuous and Discrete Behaviors

The simple way in which the basic definitions of discrete systems are extended to continuous systems may make us naively think that their behavior cannot be very different, provided the marking is "large enough". We will see in this section that this is not completely true.

For example, look at the system in Figure 1. Each time t_1 and t_2 are fired in their maximal enabling degree, the marking of p_1 is cut by half. Thus, we can always find a marking such that for every successor the marking of p_1 is as small as required. But it never reaches zero (remember the Zenon's paradox). Therefore, in the continuous system we can always go on firing transitions t_1 and t_2, while that is clearly not true in the discrete system, no matter how big the initial marking is.

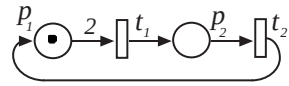


Fig. 1. A non str. live discrete system that never blocks if it is seen as continuous.

Nevertheless, the enabling degree of these continuous transitions decreases with each firing, and it could be thought that the continuous behavior is not that different from the discrete one: in the end a marking will be reached such that for every successor the enabling degree of both transitions is "almost zero". But this is a very simple system, and things may get much more entangled. Observe the system in Figure 2 (a). If we analyze this net as a discrete P/T net,

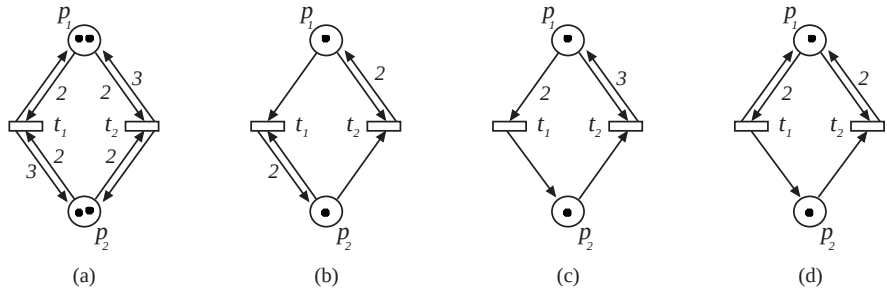


Fig. 2. Four bounded and strongly connected systems are which behave in a very different way if they are considered as discrete or as continuous:
(a) is non str. live, ϵ-live, non str. lim-live;
(b) is non str. live, non ϵ-live, non str. lim-live;
(c) is live, non str. ϵ-live, non str. lim-live;
(d) is live, ϵ-live, non str. lim-live.

it is not str. live. For any initial marking, t_1 or t_2 can be fired sufficiently many times to reach a marking in which p_1 or p_2 are marked with just one token, which is clearly a deadlock. On the other hand, the continuous system displays a completely different behavior. For instance, with the given initial marking, firing the sequence $t_1 \frac{1}{2} t_1 \frac{1}{4} t_1 \frac{1}{8} t_1 \ldots$ we can obtain a marking such that the enabling of t_1 and t_2 is as small as desired (observe that the marking of p_1 decreases exponentially). However, and unlike the previous example, the marking of p_1 is not unavoidably led to zero, since it can be increased again firing t_2.

The reason for the completely different behavior of the discrete and the continuous system in this case is that the continuous system considers the tokens as composed of *infinitely many* parts, and hence, we do not find the restriction that in the discrete system leads to not being able to redistribute the tokens. In other words, the "problem" is that the gap between natural numbers is discrete, and thus any decreasing sequence of natural numbers eventually reaches a lower bound, in which it may get blocked. This is not true in general for sequences of rational/real numbers.

This example shows also that systems that might be considered "equivalent" as discrete systems (their reachability graphs are isomorphic) may not be so if they are seen as continuous systems. From a discrete systems perspective, there is no difference between systems (a) and (b) in Figure 2. (In particular, the token-

flow matrix of both systems is the same, although the **Pre** and **Post** matrices are different.) However, their behaviors as continuous systems are completely different. The system on the right follows the behavior of the discrete system: for any initial marking firing t_1 or t_2 in a large enough amount we reach a deadlock. On the contrary, we have seen that the system on the left never deadlocks with the firing of a finite sequence.

It may also happen that a system deadlocks if it is seen as continuous and does not deadlock as discrete. Look for instance at the system in Figure 2 (c). As a discrete system, it is live with the given marking. However for any initial marking, $\mathbf{m_0}$, the firing of t_1 in an amount of $\mathbf{m_0}[p_1]/2$ in the continuous system leads to $[0, \mathbf{m_0}[p_1]/2 + \mathbf{m_0}[p_2]$ and the system gets blocked.

On the other hand, the system in Figure 2 (d), which as discrete has a reachability graph isomorphic to the one of the system in Figure 2 (c), as continuous never reaches a marking with no transition enabled. Observe that although these two systems have the same behavior as discrete, there is a big difference in the underlying nets: the system in Figure 2 (c) is non live with any initial marking with an even number of tokens in p_1; while any marking greater than or equal to $[1, 1]$ makes live the system in Figure 2 (d). It is clear that a system cannot be live as continuous if its liveness as discrete relies strongly on the particular marking. That is, a system that is live as discrete with a certain marking, but not live with a multiple of it, cannot be live as continuous. Scaling liveness monotonicity, which may be desirable, but is not compulsory in discrete systems, appears a basic property if we want to study them as continuous systems.

With the different systems in Figure 2, we have seen that, even in the case of EQ nets, the behaviors of a system, if it is considered as discrete or continuous, do not necessarily coincide. We wonder whether in simpler classes, such as live and bounded FC or CF systems, discrete and continuous behaviors are analogous. Let us consider a basic property of discrete bounded systems: no infinite firing sequence exists in which the markings are all different. Two example systems, one FC and the other CF, both live, bounded and reversible (the initial marking can always be returned to), are shown in Figure 3. In both, even this simple property is violated. For the FC system on the left, all the intermediate markings when the sequence $t_3 t_4 t_7 \frac{1}{2} t_1 \frac{1}{2} t_2 \frac{1}{2} t_4 \frac{1}{2} t_5 \frac{1}{2} t_7 \frac{1}{4} t_1 \frac{1}{4} t_2 \frac{1}{4} t_4 \frac{1}{4} t_5 \frac{1}{4} t_7 \ldots \frac{1}{2^k} t_1 \frac{1}{2^k} t_2 \frac{1}{2^k} t_4 \frac{1}{2^k} t_5 \frac{1}{2^k} t_7 \ldots$ is fired, are different. The same happens to the CF system on the right if we fire the sequence $t_1 \frac{1}{2} t_3 \frac{1}{2} t_1 \frac{1}{4} t_3 \frac{1}{4} t_1 \ldots$. This is completely different from what happens in discrete systems.

4 A New Concept: Limit Reachability

Let us go back to the system in Figure 2 (a). We have seen that a state can be reached in which the marking of p_1 is as small as desired. For some applications, it might be reasonable to consider that we can reach a marking such that this place does not contain any token. In other words, to include the marking that would be obtained in the limit as a reachable marking.

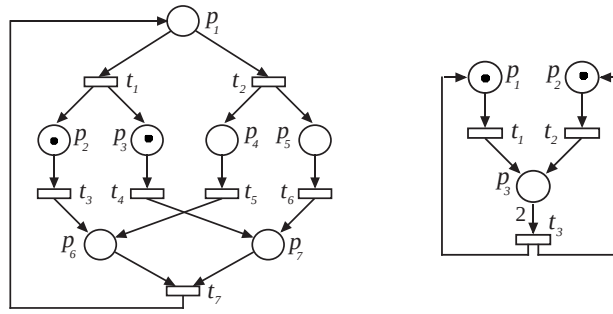


Fig. 3. Two reversible, live and bounded discrete P/T systems. When considered as continuous systems, infinitely long firing sequences exist in which the transitions are fired in their maximal enabling degree, and such that every intermediate marking appears only once.

Definition 6. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a continuous system. We say that a marking* $\mathbf{m} \in (\mathbb{R}^+ \cup \{0\})^{|P|}$ *is* limit reachable *iff a sequence of reachable markings* $\{\mathbf{m}_i\}_{i \geq 1}$ *exists verifying*

$$\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}_1 \xrightarrow{\sigma_2} \mathbf{m}_2 \cdots \mathbf{m}_{i-1} \xrightarrow{\sigma_i} \mathbf{m}_i \cdots$$

and $\lim_{i \to \infty} \mathbf{m}_i = \mathbf{m}$.

The firing sequence may be null after a finite number of firings, therefore the reachable markings are in particular limit reachable.

The limit reachability space*,* $\text{lim-RS}_C(\mathcal{N}, \mathbf{m_0})$*, is the set of limit reachable (and in particular reachable) markings.*

The definition of boundedness does not change with the new concept of limit reachability (if every $\mathbf{m}_i \leq \mathbf{k}$, then $\lim_{i \to \infty} \mathbf{m}_i \leq \mathbf{k}$).

There is a strong relationship between the LRS_C and the lim-RS_C of a continuous system. In fact, they coincide in consistent systems in which every transition is fireable.

Theorem 7. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be consistent and such that each transition can be fired at least once.*

Then $\text{lim-RS}_C(\mathcal{N}, \mathbf{m_0}) = \text{LRS}_\text{C}(\mathcal{N}, \mathbf{m_0})$.

Proof. It is clear that $\text{lim-RS}_C(\mathcal{N}, \mathbf{m_0}) \subseteq \text{LRS}_\text{C}(\mathcal{N}, \mathbf{m_0})$, since $\text{LRS}_\text{C}(\mathcal{N}, \mathbf{m_0})$ is a closed set that includes $\text{RS}_C(\mathcal{N}, \mathbf{m_0})$.

For the "$\supseteq$", let $\mathbf{m} \in \text{LRS}_\text{C}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$. Applying Proposition 4, we have that from $\mathbf{m_0}$ a positive marking $\mathbf{m'} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma'}$ can be reached. We will prove that $\mathbf{m} \in \text{lim-RS}_C(\mathcal{N}, \mathbf{m'})$. Observe that $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} = \mathbf{m'} + \mathbf{C} \cdot (\boldsymbol{\sigma} - \boldsymbol{\sigma'})$. Being $\mathcal{N}$ consistent, a T-semiflow, $\mathbf{x}$, exists such that $\mathbf{x} + \boldsymbol{\sigma} - \boldsymbol{\sigma'} \geq \mathbf{0}$, and thus $\mathbf{m} = \mathbf{m'} + \mathbf{C} \cdot (\boldsymbol{\sigma} - \boldsymbol{\sigma'} + \mathbf{x})$. Since $\mathbf{m'} > \mathbf{0}$, α and σ'' exist such that σ'' is fireable from $\mathbf{m'}$ and $\boldsymbol{\sigma''} = \alpha(\boldsymbol{\sigma} - \boldsymbol{\sigma'} + \mathbf{x})$, i.e., a sequence proportional to the vector leading from $\mathbf{m'}$ to $\mathbf{m}$ can be fired. If $\alpha \geq 1$, it is clear that $\mathbf{m}$

can be reached from $\mathbf{m}'$. Otherwise, the firing of σ'' leads to $\mathbf{m}' + \mathbf{C} \cdot \boldsymbol{\sigma}'' = \mathbf{m}' + \alpha\mathbf{C} \cdot (\boldsymbol{\sigma} - \boldsymbol{\sigma}' + \mathbf{x}) = \alpha\mathbf{m} + (1-\alpha)\mathbf{m}'$, i.e.,

$$\mathbf{m_0} \xrightarrow{\sigma'} \mathbf{m}' \xrightarrow{\sigma''} \alpha\mathbf{m} + (1-\alpha)\mathbf{m}'$$

Clearly, if σ'' was fireable from $\mathbf{m}'$, $(1-\alpha)\,\sigma''$ is fireable from $(1-\alpha)\mathbf{m}'$. Hence

$$\alpha\mathbf{m} + (1-\alpha)\mathbf{m}' \xrightarrow{(1-\alpha)\,\sigma''} \alpha\mathbf{m} + \alpha(1-\alpha)\mathbf{m} + (1-\alpha)^2\mathbf{m}'$$

Repeating the procedure we build a sequence of markings whose limit is $\mathbf{m}$. □

This does not mean that for any vector $\mathbf{x}$ such that $\mathbf{m_0} + \mathbf{C} \cdot \mathbf{x} \geq \mathbf{0}$ a sequence with this firing vector is enabled at $\mathbf{m_0}$. For instance, in the example of Figure 4, $[0,1,0,0,1] = [1,0,0,0,1] + \mathbf{C} \cdot [0,1,1,0,0]^T$ is reachable, but no sequence with firing vector $[0,1,1,0,0]$ is enabled.

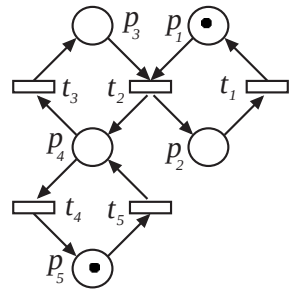


Fig. 4. In this system $[0,1,0,0,1]$ is a reachable marking, although no enabled sequence has $[0,1,1,0,0]$ as its firing vector.

The equality of the $\mathrm{LRS_C}$ and the lim-RS_C does not hold in general if the system is not consistent or not every transition is fireable. For instance, in the system on the left in Figure 5, which is consistent, but in which no transition can be fired, the marking $[0,1,0,0]$ belongs to the $\mathrm{LRS_C}$ but not to the lim-RS_C. The same happens to the marking $[0,1,0,0,1]$ in the system on the right. In this system every transition can be fired, but it is not consistent. For the moment nothing can be said about the complexity of computing the lim-RS_C in the general case, not even whether it is decidable or not. However, the setting in which the equality holds seems general enough to cover many interesting cases.

5 On Liveness Analysis

Two of the main properties we have been discussing about all along this work are liveness and deadlock-freeness. In this section we will present two possible extensions of the definitions of liveness to continuous P/T systems, one w.r.t.

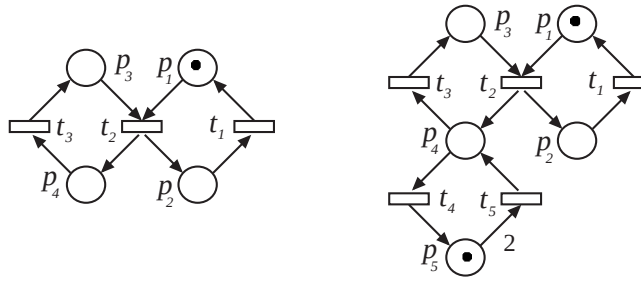


Fig. 5. Two continuous systems for which either not every transition is fireable (left) or the net is not consistent (right).

the $\mathrm{RS_C}$ (ϵ-*liveness*) and the other w.r.t. the lim-RS_C (lim-*liveness*), and the corresponding two definitions of deadlock-freeness. We will compare these two liveness definitions and also relate them to discrete liveness. This will allow to deduce that necessary conditions for discrete liveness are necessary for lim-liveness too. Then, we will concentrate on the classes of EQ and FC nets. We will see that for strongly connected and str. bounded EQ systems str. lim-liveness and (discrete) str. liveness coincide, and they are also equivalent to str. ϵ-liveness for FC nets.

5.1 Liveness Definitions

Let us start defining liveness w.r.t. the $\mathrm{RS_C}$. A naive generalization of the discrete definition, following the approach used to define boundedness, leads to the following statement: a transition t of a continuous P/T system is live iff from every reachable marking, $\mathbf{m}$, another marking can be reached, $\mathbf{m'}$, at which the transition is enabled, i.e., $\mathrm{enab}(t, \mathbf{m'}) > 0$.

According to this definition, transition t_1 in Figure 1 is live, since for every reachable marking a successor exists such that the marking of p_1 is greater than zero. From our discrete-biased point of view, this does not seem to be what one would desire of a live transition. Therefore, let us try to modify the definition to avoid this kind of behavior.

The problem in this example is that with a finite number of firings the marking of p_1 and p_2 can be done indefinitely small, but not zero. With the idea of not allowing this to be considered live, we introduce an improved version of the definition of liveness:

Definition 8. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous P/T system. A transition t is* ϵ-live *iff $\epsilon > 0$ exists such that for any reachable marking, $\mathbf{m}$, a successor $\mathbf{m'} \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m})$ can be found such that $\mathrm{enab}(t, \mathbf{m'}) \geq \epsilon$.*

So, if the enabling of a transition can be made as small as desired and never grows back, this transition is not ϵ-live. As in discrete systems, we will say that a continuous system is str. ϵ-live if a marking $\mathbf{m_0}$ exists such that in $\langle \mathcal{N}, \mathbf{m_0} \rangle$

every transition is ϵ-live. For instance, the system in Figure 2 (a) is ϵ-live with the given marking.

In an analogous way we can define deadlock-freeness.

Definition 9. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous P/T system. It* ϵ-deadlocks *iff for every $\epsilon > 0$ a reachable marking, $\mathbf{m}_\epsilon$, exists such that for every successor $\mathbf{m}'_\epsilon \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m}_\epsilon)$, and every transition t, $\mathrm{enab}(t, \mathbf{m}'_\epsilon) < \epsilon$.*

Clearly, the system in Figure 1 ϵ-deadlocks.

Another possibility is to define liveness w.r.t. the lim-RS_C. With this definition of reachability, the extension of liveness is immediate:

Definition 10. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous P/T system. A transition t is* lim-live *iff for any marking $\mathbf{m} \in \text{lim-}\mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$ a successor $\mathbf{m}' \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m})$ exists such that $\mathrm{enab}(t, \mathbf{m}') > 0$.*

In other words, a transition is non lim-live iff a sequence of successively reachable markings exists which converges to a marking such that none of its successors enables the transition. Observe that none of the systems in Figure 2 is lim-live. For example, in the system in 2 (a), firing once either t_1 or t_2 a deadlock is reached. For the system in 2 (b), take for instance the following sequence of markings:

$$\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{\frac{1}{2}t_1} \mathbf{m}_2 \xrightarrow{\frac{1}{4}t_1} \mathbf{m}_3 \xrightarrow{\frac{1}{8}t_1} \dots$$

That is, $\mathbf{m}_k = [1/2^{(k-1)}, 4 - 1/2^{(k-1)}]$. Clearly, a limit of this sequence exists, $\mathbf{m} = [0, 4]$, and no transition is enabled there. Hence, the system is not lim-live. Even more, it can be proven that no marking makes this system lim-live. Therefore, we can say that the continuous P/T net is not str. lim-live.

The following properties of bounded lim-live systems can be immediately deduced.

Theorem 11. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a consistent, bounded and* lim-*live continuous system. Then,*

1. $\mathrm{LRS_C}(\mathcal{N},\mathbf{m_0})$=lim-$\mathrm{RS_C}(N,\mathbf{m_0})$*, i.e. there is no spurious solution of the state equation.*
2. *$\mathcal{N}$ is str. bounded.*
3. *$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is reversible w.r.t. the* lim-RS_C*.*

Proof. For (1), since the system is lim-live, a fireable sequence exists that contains all the transitions. Then, applying Theorem 7, the result is proved.

(2) is immediate from Theorem 5.

To prove (3), let $\mathbf{m} \in \text{lim-}\mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$. Observe that, since the net is consistent, $\mathbf{m_0} \in \mathrm{LRS_C}(\mathcal{N}, \mathbf{m})$. Then, applying (1) to $\langle \mathcal{N}, \mathbf{m} \rangle$, we obtain that $\mathbf{m_0} \in \text{lim-}\mathrm{RS_C}(\mathcal{N}, \mathbf{m})$. □

Analogously, we can define lim-deadlock:

Definition 12. *A continuous P/T system $\langle \mathcal{N}, \mathbf{m_0} \rangle$* lim-deadlocks *iff a marking $\mathbf{m} \in \text{lim-}\mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$ exists such that $\mathrm{enab}(t, \mathbf{m}) = 0$ for every transition t.*

For instance, all the systems in Figure 2 lim-deadlock.

5.2 Liveness Monotonicity

In discrete systems it may happen that a system is live with a certain marking, and non-live with any other marking, in particular with a multiple marking. In continuous systems an ϵ-live (lim-live) system is also ϵ-live (lim-live) with any multiple/fraction of the initial marking. That is, ϵ-liveness and lim-liveness are monotonic w.r.t. scaling of the marking.

Proposition 13. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a continuous P/T system. If it is ϵ-live (*lim*-live), then for every $\alpha > 0$ $\langle \mathcal{N}, \alpha\mathbf{m_0} \rangle$ is ϵ-live (*lim*-live).*

Proof. Assume $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is ϵ-live and $\alpha > 0$ exists such that $\langle \mathcal{N}, \alpha\mathbf{m_0} \rangle$ is not ϵ-live. Since $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is ϵ-live, a constant $\epsilon > 0$ exists such that for every $\mathbf{m} \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$, and every transition t, a successor of $\mathbf{m}$ exists for which the enabling of t is greater than ϵ. $\langle \mathcal{N}, \alpha\mathbf{m_0} \rangle$ is not ϵ-live, therefore a transition t' and a sequence σ' exist such that $\alpha\mathbf{m_0} \xrightarrow{\sigma'} \mathbf{m'}$, and for every successor of $\mathbf{m'}$ the enabling of t' is less than $\alpha\epsilon$. Observe that $\frac{1}{\alpha}\sigma'$ can be fired at $\mathbf{m_0}$, i.e., $\mathbf{m_0} \xrightarrow{\frac{1}{\alpha}\sigma'} \frac{1}{\alpha}\mathbf{m'}$. A successor of $\frac{1}{\alpha}\mathbf{m'}$ exists such that the enabling of t' is greater than ϵ. Hence, a successor of $\mathbf{m'}$ exists for which the enabling of t' is greater than $\alpha\epsilon$, contradiction.

An analogous proof can be used in the case of lim-liveness. □

A question that naturally arises is whether discrete liveness monotonicity is a stronger result than ϵ- or lim-liveness. This is false in the case of lim-liveness. For instance, the system in Figure 2 (d) is live as a discrete system with the given marking or a larger one. But it is not str. lim-live, since for any initial marking firing $\frac{\mathbf{m_0}[p_1]}{2} t_2 \frac{\mathbf{m_0}[p_1]}{4} t_2 \frac{\mathbf{m_0}[p_1]}{8} t_2 \ldots$, in the limit a marking is reached in which no transition is enabled. However, it is true w.r.t. ϵ-liveness, that is, discrete liveness monotonicity implies ϵ-liveness.

Theorem 14. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a consistent system which is live as discrete with any marking multiple of the initial one. Then, it is ϵ-live.*

Proof. Consider $\langle \mathcal{N}, \mathbf{m_0} \rangle$ as a continuous system. The proof will be done in two steps. First, assume a reachable marking $\mathbf{m}$ and a transition t exist such that no successor of $\mathbf{m}$ enables t. Taking a constant k big enough, a discrete sequence can be fired from $k\mathbf{m_0}$ leading to a marking which is as close as desired to $k\mathbf{m}$ (it may be exact if in the sequence leading to $\mathbf{m}$ all the transitions are fired in a rational amount). Since the discrete system is live, a successor of this marking enables t. Hence, a successor of $\mathbf{m}$ exists that enables t, contradiction.

Therefore, for every transition and every marking a successor exists in which this transition is enabled. It might happen that the system were not ϵ-live because the enabling approached zero, i.e., for every $\epsilon > 0$, t and $\mathbf{m}$ exist such that for every successor the enabling of t is less than ϵ. Since the system is consistent, reasoning as in Theorem 7, we can reach a marking that is as close as desired to any solution of the $\mathrm{LRS_C}$, in particular a making close to $\mathbf{m_0}$. $\langle N, \mathbf{m_0} \rangle$ is live as discrete, hence t can be fired again in a "big amount". □

5.3 Relations between Discrete Liveness, ϵ-Liveness, and lim-Liveness

In Section 3 we compared the properties of a system as discrete with its properties as continuous, this last one interpreted using the immediate continuous extension of reachability. The examples that appear there show in particular that a system can be str. live as discrete and not str. ϵ-live (Figure 2 (c)); and the reverse, it can be str. ϵ-live and not str. live (Figure 2 (a)).

The concept of limit reachability was then introduced, trying to bring the continuous properties nearer to what one would expect. As we have seen, lim-liveness cannot be deduced from (discrete) str. liveness (Figure 2 (c)). However, any lim-live system is str. live if it is seen as discrete. (Although not necessarily live, i.e., the structure of the net is "correct", although the marking may be "not large enough".)

Theorem 15. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a bounded* lim*-live P/T system. Then, $\mathcal{N}$ is str. live and str. bounded as a discrete net.*

Proof. Assume $\mathcal{N}$ is not str. live as a discrete net. We will see that we can find a sequence of successively reachable markings in the continuous system, such that at the limit at least one transition is disabled, which contradicts lim-liveness.

$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is not live as a discrete system, therefore a sequence σ_1 and a transition t_{j_1} exist such that $\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}_1$ and for every successor of $\mathbf{m}_1$ t_{j_1} is disabled.

Take now $\langle \mathcal{N}, 2\mathbf{m}_1 \rangle$. It is not live as a discrete system, therefore a sequence σ_2 and a transition t_{j_2} exist such that $2\mathbf{m}_1 \xrightarrow{\sigma_2} \mathbf{m}_2$ and for every successor of $\mathbf{m}_2$ t_{j_2} is disabled.

Analogously, $\langle \mathcal{N}, 2\mathbf{m}_2 \rangle$ is not live as a discrete system ...

Repeating this procedure, a sequence of markings of the continuous system is obtained:

$$\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}_1 \xrightarrow{\frac{1}{2}\sigma_2} \frac{1}{2}\mathbf{m}_2 \xrightarrow{\frac{1}{4}\sigma_3} \frac{1}{4}\mathbf{m}_3 \longrightarrow \ldots \xrightarrow{\frac{1}{2^{k-1}}\sigma_k} \frac{1}{2^{k-1}}\mathbf{m}_k \longrightarrow \ldots$$

For simplicity, let us denote $\mathbf{m}'_k = \frac{1}{2^{k-1}}\mathbf{m}_k$ and $\sigma'_k = \frac{1}{2^{k-1}}\sigma_k$. Then,

$$\mathbf{m_0} \xrightarrow{\sigma'_1} \mathbf{m}'_1 \xrightarrow{\sigma'_2} \mathbf{m}'_2 \longrightarrow \ldots \xrightarrow{\sigma'_k} \mathbf{m}'_k \longrightarrow \ldots$$

The number of transitions is finite, therefore an infinite subsequence exists such that the disabled transition is always the same. We will denote this transition as t.

Since the system is bounded, a convergent subsequence of the former subsequence exists (by Bolzano-Weierstrass Theorem, any bounded sequence contains a convergent subsequence). We will denote this latter subsequence as $\{\mathbf{m}'_{i_k}\}_{k \geq 1}$.

That is,

$$\mathbf{m_0} \xrightarrow{\sigma''_{i_1}} \mathbf{m}'_{i_1} \xrightarrow{\sigma''_{i_2}} \mathbf{m}'_{i_2} \longrightarrow \ldots \xrightarrow{\sigma''_{i_k}} \mathbf{m}'_{i_k} \longrightarrow \ldots$$

where $\lim_{k\to\infty} \mathbf{m}'_{i_k} = \mathbf{m}'$.

$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is lim-live, therefore, $\alpha > 0$, and a firing sequence σ exist such that

$$\mathbf{m}' \xrightarrow{\sigma} \widetilde{\mathbf{m}}' \xrightarrow{\alpha t}$$

Let $\epsilon = \frac{1}{2} \min_{p\in P}\{\mathbf{m}'[p] \mid \mathbf{m}'[p] > 0\}$. Since $\mathbf{m}' = \lim_{k\to\infty} \mathbf{m}'_{i_k}$, a certain k_0 exists such that for every $k \geq k_0$ and every place p, $|\mathbf{m}'_{i_k}[p] - \mathbf{m}'[p]| < \epsilon$. Thus, for every $k \geq k_0$, $\mathbf{m}'_{i_k} \geq \frac{1}{2}\mathbf{m}'$ and

$$\mathbf{m}'_{i_k} \xrightarrow{\frac{1}{2}\sigma} \widetilde{\mathbf{m}}'_{i_k} \xrightarrow{\frac{\alpha}{2}t}$$

Let $K_i = 2^{i_k - 1}$. Then, $\mathbf{m}_{i_k} = K_i \mathbf{m}'_{i_k} \xrightarrow{\frac{K_i}{2}\sigma} K_i \widetilde{\mathbf{m}}'_{i_k} \xrightarrow{\frac{K_i \alpha}{2} t}$, and taking k big enough, an integer sequence that enables t can be fired from $\mathbf{m}_{i_k}$, contradiction. □

With respect to the relationship between lim-liveness and ϵ-liveness, a similar result can be proven, i.e., a bounded continuous lim-live system is ϵ-live. The reverse is not true, as the system in Figure 2 (a) shows.

Theorem 16. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a bounded* lim*-live P/T system. Then, $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is ϵ-live.*

Proof. Assume $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is not ϵ-live. Then, for every $k > 0$ a transition t_{j_k} and a marking $\mathbf{m}_k \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m_0})$ exist such that for every $\widetilde{\mathbf{m}}_k \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m}_k)$, $\mathrm{enab}(t_{j_k}, \widetilde{\mathbf{m}}_k) < 1/k$. Since the number of transitions is finite we can assume w.l.o.g. that all the t_{j_k} coincide. We will denote this transition as t.

Observe that $\mathbf{m}_1$ is reachable from $\mathbf{m_0}$, but nothing ensures that $\mathbf{m}_2$ can be reached from $\mathbf{m}_1$. By Theorem 15, this system is str. live and str. bounded, hence consistent. Thus, $\mathbf{m}_2 \in \mathrm{LRS_C}(\mathcal{N}, \mathbf{m}_1)$. Moreover, being lim-live, every transition is fireable, and applying Theorem 7, $\mathbf{m}_2 \in \text{lim-}\mathrm{RS}_C(\mathcal{N}, \mathbf{m}_1)$. Therefore $\mathbf{m}'_2 \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m}_1)$ exists such that $|\mathbf{m}'_2 - \mathbf{m}_2| < 1/2$. Analogously, $\mathbf{m}_3 \in \text{lim-}\mathrm{RS}_C(\mathcal{N}, \mathbf{m}'_2)$, and repeating the reasoning $\mathbf{m}'_3 \in \mathrm{RS_C}(\mathcal{N}, \mathbf{m}'_2)$ exists such that $|\mathbf{m}'_3 - \mathbf{m}_3| < 1/3$. In general,

$$\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}_1 \xrightarrow{\sigma'_2} \mathbf{m}'_2 \xrightarrow{\sigma'_3} \mathbf{m}'_3 \ldots \xrightarrow{\sigma'_k} \mathbf{m}'_k \ldots$$

and $|\mathbf{m}_k - \mathbf{m}'_k| < 1/k$. This defines a bounded sequence of markings, therefore, applying the Bolzano-Weierstrass Theorem, a convergent subsequence $\{\mathbf{m}'_{i_k}\}_{k>0}$ exists. Let $\mathbf{m}' = \lim_{k\to\infty} \mathbf{m}'_{i_k}$.

$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is lim-live and $\mathbf{m}' \in \text{lim-}\mathrm{RS}_C(\mathcal{N}, \mathbf{m_0})$, hence a (finite) sequence σ and $\alpha > 0$ exist such that

$$\mathbf{m}' \xrightarrow{\sigma} \widetilde{\mathbf{m}}' \xrightarrow{\alpha t}$$

Define $\epsilon = \frac{1}{2} \min\{\mathbf{m}'[p] \mid \mathbf{m}'[p] > 0\}$. Then, applying the limit definition and the way the sequence has been built, a certain k_0 exists such that for every $k \geq k_0$,

$|\mathbf{m}'_{i_k} - \mathbf{m}'| < \frac{1}{2}\epsilon$, and $|\mathbf{m}'_{i_k} - \mathbf{m}_{i_k}| < \frac{1}{2}\epsilon$. Thus, $|\mathbf{m}_{i_k} - \mathbf{m}'| < \epsilon$ and by definition of ϵ, $\mathbf{m}_{i_k} \geq 1/2\,\mathbf{m}'$. Therefore,

$$\mathbf{m}_{i_k} \xrightarrow{\frac{1}{2}\sigma} \widetilde{\mathbf{m}}_{i_k} \xrightarrow{\frac{\alpha}{2}t}$$

If k is big enough, $\frac{\alpha}{2} > \frac{1}{i_k}$. Contradiction, since for every successor of $\mathbf{m}_{i_k}$ the enabling of t is less than $1/i_k$. □

We have seen that a system may be str. discrete live (Figure 2 (a)) or str. ϵ-live (Figure 2 (c)) and not str. lim-live. We might think that if both conditions were required, i.e., the system is str. ϵ-live as a continuous system and str. live as a discrete one, perhaps str. lim-liveness could be deduced. Actually, this is not the case, as can be observed in the system in Figure 2 (d). The problem in this example is that there are solutions of the state equation, that cannot be reached in the discrete system but are reachable at the limit, which correspond to deadlocks. For example, with the given initial marking, firing the sequence $\frac{1}{2}t_1\frac{1}{4}t_1\frac{1}{8}t_1\ldots$, we reach in the limit the marking $[0, 2]$, that clearly is a deadlock.

The results in Theorem 15 and Theorem 16, and the counterexamples in Figure 2 are summarized in the diagram at Figure 6. In Subsection 5.5 we will see that these results can be improved if we restrict to selected subclasses.

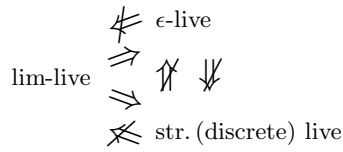


Fig. 6. Relationships among lim-liveness, ϵ-liveness and discrete liveness for general P/T nets.

5.4 Two Necessary Conditions for lim-Liveness

From Theorem 15 it is clear that any necessary condition for a discrete system to be str. live and str. bounded, is also necessary for it to be str. lim-live and bounded. In particular the rank theorem (see [9] for a recent survey) is a necessary condition based on the existence of left and right annullers of the token-flow matrix, and the existence of an upper bound on the rank of this matrix, which is the number of *equal conflict sets*. Two transitions, t and t', are said to be in *equal conflict* (EQ) relation when $\mathbf{Pre}[P, t] = \mathbf{Pre}[P, t'] \neq \mathbf{0}$. This is an equivalence relation and the set of all the equal conflict sets is denoted by SEQS.

Theorem 17. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a* lim*-live and bounded continuous system. Then,* $\mathcal{N}$ *is consistent, conservative and* $\text{rank}(\mathbf{C}) \leq |\text{SEQS}| - 1$.

Other structural elements that are useful in the analysis of lim-liveness are siphons. A siphon is a set of places, P'', such that ${}^\bullet P'' \subseteq P''^\bullet$. Observe that an empty siphon cannot be marked. Hence, a necessary condition for lim-liveness is that no marking can be reached in which a siphon is empty. In [7] the conditions for a set of places being a siphon were stated as the solutions to a set of linear inequalities: A set $\Sigma \subseteq P$ is a siphon of $\mathcal{N}$ iff $\mathbf{y} \geq \mathbf{0}$ exists such that $\|\mathbf{y}\| = \Sigma$ and $\mathbf{y} \cdot \mathbf{C}_\Sigma \geq \mathbf{0}$, where $\mathcal{N}_\Sigma = \langle P, T, \mathbf{Pre}_\Sigma, \mathbf{Post} \rangle$ is such that $\mathbf{Pre}_\Sigma[p,t] = 0$ iff $\mathbf{Pre}[p,t] = 0$, and $\mathbf{Pre}_\Sigma[p,t] \geq \sum_{p' \in t^\bullet} \mathbf{Post}[p',t]$, otherwise. If lim-RS$_C$ = LRS$_\mathrm{C}$, the absence of a marking in which a siphon is not marked can be checked using a system of linear inequalities:

Theorem 18. *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a consistent P/T continuous system. If a solution of the following system of inequalities exists, the system is not* lim*-live.*

$$\begin{aligned} \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} &\geq \mathbf{0} \\ \mathbf{y} \cdot \mathbf{C}_\Sigma &\geq \mathbf{0} \\ \mathbf{y} \cdot \mathbf{m} &= 0 \\ \mathbf{y} &\geq \mathbf{0} \\ \boldsymbol{\sigma} &\geq \mathbf{0} \end{aligned}$$

In discrete systems there exists a symmetry between traps and siphons (a trap is a set of places, P', such that $P'^\bullet \subseteq {}^\bullet P'$), in the sense that marked traps cannot be emptied and empty siphons cannot be marked. This symmetry is lost in continuous systems if lim-reachability is considered, because although an empty siphon cannot become marked, a trap can be emptied. For instance, in Figure 7, $\{p_1, p_2, p_3, p_4\}$ is a trap, that is emptied by the firing of $t_1 t_2 \ \frac{1}{2} t_3 \frac{1}{2} t_4 \frac{1}{2} t_2 \ \frac{1}{4} t_3 \frac{1}{4} t_4 \frac{1}{4} t_2 \ \frac{1}{8} t_3 \frac{1}{8} t_4 \frac{1}{8} t_2 \ldots$ This means that traps cannot be used to improve the description of the reachability conditions given by the state equation as in discrete systems [12].

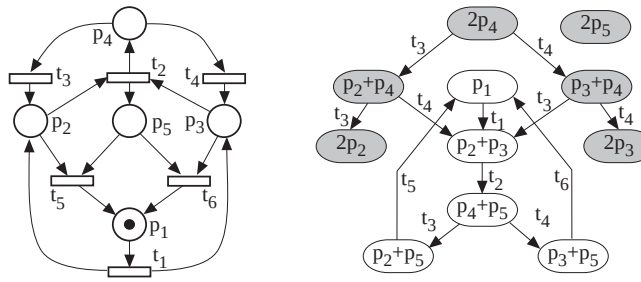


Fig. 7. A live system and its LRG.

5.5 Particular Results for Some Subclasses

As usual, the results obtained in the general frame of continuous systems can be improved if we restrict to selected subclasses. We concentrate here on two subclasses: EQ nets and FC nets.

First, it can be seen that lim-liveness and lim-deadlock freeness coincide in bounded and strongly connected EQ systems, as happened with their discrete counterparts.

Theorem 19. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a bounded, strongly connected, continuous EQ system. It is* lim*-live iff it is* lim*-deadlock-free.*

Proof. Assume $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is not lim-live. Then, a transition t and a marking $\mathbf{m} \in \text{lim-RS}_C(\mathcal{N}, \mathbf{m_0})$ exist such that for every $\mathbf{m}' \in \text{RS}_C(\mathcal{N}, \mathbf{m})$ the enabling of t is zero. Let $p \in {}^\bullet t$. All the transitions in $p^\bullet$ are in equal conflict relation, hence none of the output transitions of p fires again. The system is bounded, therefore a sequence of markings exists such that for their successors the enabling degree of the input transitions of p converges to zero. Applying the Bolzano-Weierstrass theorem, a convergent subsequence of markings exists. Neither t, nor any input transition of p is enabled at the marking reached in the limit, and they will never be enabled again. Repeating the reasoning, since the net is strongly connected, and the number of transitions is finite, we finally reach a marking in which no transition is enabled. □

In discrete EQ systems the rank theorem is a characterization of str. liveness and str. boundedness [13]. For continuous EQ systems this result can be improved: a characterization of lim-liveness and boundedness can be obtained analogous to the one that exists for liveness and boundedness of discrete FC systems [6]. This provides a simple, polynomial time, way to prove lim-liveness of EQ systems:

Theorem 20. *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a continuous EQ system. The following conditions are equivalent:*

1. *The system is* lim*-live and bounded.*
2. *The system is consistent, conservative,* $\text{rank}(\mathbf{C}) = |\text{SEQS}| - 1$ *(or, equivalently, it is str. bounded and str. live as discrete) and the support of every P-semiflow is marked, i.e.,* $\nexists\, \mathbf{y} \geq \mathbf{0}$ *such that* $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, $\mathbf{y} \cdot \mathbf{m_0} = 0$.

Proof. For "1⇒2", applying Theorem 15, and the characterization of str. liveness and str. boundedness for EQ nets [13], the net must be consistent, conservative and $\text{rank}(\mathbf{C}) = |\text{SEQS}| - 1$. Assume a P-semiflow, $\mathbf{y}$, exists that is not marked. Then, for every reachable marking, $\mathbf{m}$, $\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0} = 0$, i.e., none of the places in $\|\mathbf{y}\|$ can ever be marked. Hence, their output transitions cannot be fired, contradiction.

For "2⇒1", if $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is not live, it deadlocks (Theorem 19). Let $\mathbf{m}_d \in \text{lim-RS}_C(\mathcal{N}, \mathbf{m_0})$ be a deadlock. Then, for every transition t, a place $p \in {}^\bullet t$ exists such that $\mathbf{m}_d[p] = 0$. This set of places contains the support of a P-semiflow [13], and it is not marked, contradiction. □

The properties of EQ systems allow to extend to lim-liveness and boundedness a sufficient condition for (discrete) str. liveness and str. boundedness [9]. The idea is to transform the system into an EQ system, and apply the results of this class. More specifically, each coupled conflict set is transformed into an EQ set. The coupled conflict relation is defined as the transitive closure of the str. conflict relation, where t and t' are in str. conflict relation iff ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. The set of all the equivalence classes is denoted by SCCS. We skip the proof, since it is analogous to the one given in [9] for the discrete case.

Theorem 21. *Let $\mathcal{N}$ be consistent, conservative and* $\text{rank}(\mathbf{C}) = |\text{SCCS}| - 1$. *Then, $\mathcal{N}$ is str.* lim*-live as a continuous net. Moreover, any marking that marks every P-semiflow makes the system* lim*-live.*

In strongly connected str. bounded EQ systems str. lim-liveness is equivalent to (discrete) str. liveness, though not in general to str. ϵ-liveness (see Figure 2 (a)). For strongly connected str. bounded FC nets a stronger result holds: (discrete) str. liveness, str. ϵ-liveness, and str. lim-liveness are equivalent.

Theorem 22. *Let $\mathcal{N}$ be a strongly connected, str. bounded FC net. The following conditions are equivalent:*

1. *$\mathcal{N}$ is str.* lim*-live.*
2. *$\mathcal{N}$ is str. ϵ-live*
3. *$\mathcal{N}$ is str. (discrete) live.*

Proof. "(1)⇒(2)" is proven in Theorem 16 and "(3)⇒(1)" can be deduced from Theorem 20.

For "(2)⇒(3)", assume $\mathcal{N}$ is not str. live and let $\mathbf{m_0}$ be such that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is ϵ-live. We can assume w.l.o.g. that $\mathbf{m_0} \in \mathbb{N}^{|P|}$. The system is not live as a discrete system, hence it deadlocks, i.e., a sequence σ_d exists such that $\mathbf{m_0} \xrightarrow{\sigma_d} \mathbf{m}_d$ and no transition is enabled at $\mathbf{m}_d$. Then, for every transition t a place $p \in {}^\bullet t$ exists such that $\mathbf{m}_d[p] = 0$. The same sequence can be fired if the system is considered as continuous, and clearly it also leads to a deadlock, contradiction. □

6 Conclusions

A common practice in many fields in which systems with large state spaces appear is to relax the description by dropping integrality constraints on the state equation. Moreover, there are systems in which discrete parts are mixed with other parts, that are more naturally represented as continuous. In P/T systems this idea has led to the definition of continuous and hybrid P/T systems. Some of these models incorporate a continuous part by means of algebraic differential equations [15,2], others allow non integer markings in some places [4,14] (see [3] for a comparison of different approaches through the modelling of a benchmark example). This latter approach is the one that is considered here, although with one main difference, we study autonomous models, i.e. without any timed interpretation.

The basic definitions of P/T systems are extended to the continuous case in Section 2, allowing real non negative markings, and the firing of transitions in any real non negative amount. An immediate consequence of the possibility of firing transitions in non discrete amounts, is that the behavior of a continuous system does not change if the initial marking is scaled. This does not happen in discrete systems (for instance, a discrete system may be deadlock-free with a certain marking, and deadlock if the initial marking is doubled). This means that monotonicity of the properties w.r.t. scaling, which is not basic for the study of discrete systems, is a must if we want the continuous view to be coherent with the discrete one. In other words, a system should not be studied as continuous if the exact amount of tokens is so important to determine its behavior.

The relaxation of the notions of marking and firing, allowing positive real numbers, is quite intuitive. However, the extension of reachability is not so immediate. This concept is central for the eventual analysis of logic properties of the modeled systems, but it has not been properly investigated before. In this paper two possible definitions of reachability have been introduced, and their analysis explored.

With the first notion of reachability, the idea of a finite sequence of firings is preserved. Some examples are presented in Section 3 showing that with this definition of reachability the behavior of continuous and discrete systems can be completely different. The second notion, introduced in Section 4, allows infinitely long firing sequences, leading to the concept of limit reachable markings.

Another difference w.r.t. the discrete case is that in continuous systems the set of reachable markings is a convex set, independently of which reachability definition is considered. Apparently, deciding whether a certain marking is reachable or not, when the firing is not restricted to be integer, is more difficult than in the integer case. However, under the limit reachability definition, in most practical cases the set of reachable markings coincides with the solutions of the state equation.

Liveness and deadlock-freeness are studied in Section 5. Two definitions of liveness, ϵ-liveness and lim-liveness, and the corresponding two definitions of deadlock-freeness, have been introduced, each one associated to one of the definitions of reachability. These two continuous liveness definitions and the discrete liveness definition are compared. As a result it is deduced that in bounded systems lim-liveness implies ϵ-liveness and (discrete) str. liveness. For bounded strongly connected EQ systems, this result can be improved: str. lim-liveness and discrete str. liveness coincide. Moreover, they are also equivalent to ϵ-liveness in the case of FC systems. These relations among discrete and continuous liveness definitions allow to obtain two necessary conditions for lim-liveness and boundedness, and a sufficient one, which are analogous to the ones that exist in the discrete case.

From our experience after this work, we conclude that although both definitions of reachability are interesting, limit reachability seems to be specially convenient. On the one hand, the idea of treating very small quantities as zero is reasonable if continuous systems are considered as an approximation of discrete

systems. On the other, the simple representation of the lim-RS_C in most practical cases (the solutions of the state equation) offers a clear advantage w.r.t. the RS_C, for which there seems to be no simple way to deduce whether a certain marking can be reached or not. However, further investigations about how to extend other properties of discrete P/T systems should be done before making a choice.

References

1. H. Alla and R. David. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers*, 8(1), 1998.
2. R. Champagnat, P. Esteban, H. Pingaud, and R. Valette. Modeling hybrid systems through by means of high-level Petri nets. benefits and limitations. In M. Ferney, P. Borne, A. El Moudni, and L. Grujic, editors, *Proceedings of CIS'97*, volume 1, pages 469–474, Belfort, France, 1997.
3. R. Champagnat, H. Pingaud, H. Alla, C. Valentin-Roubinet, and J.M. Flaus. A gas storage example as a benchmark for hybrid modelling. In *Proc. of ADPM'98*, pages 123–130, Reims, France, March 1998.
4. R. David and H. Alla. Continuous Petri nets. In *Proc. of the 8th European Workshop on Application and Theory of Petri Nets*, pages 275–294, Zaragoza, Spain, 1987.
5. R. David and H. Alla. Autonomous and timed continuous Petri nets. In Rozenberg [10], pages 71–90.
6. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
7. J. Ezpeleta, J. M. Couvreur, and M. Silva. A new technique for finding a generating family of siphons, traps and ST-components. application to coloured Petri nets. In Rozenberg [10], pages 126–147.
8. D. G. Luenberger. *Introduction to Linear and non Linear Programming*. Addison-Wesley Publishing Company, 1972.
9. L. Recalde, E. Teruel, and M. Silva. On linear algebraic techniques for liveness analysis of P/T systems. *Journal of Circuits, Systems, and Computers*, 8(1):223–265, 1998.
10. G. Rozenberg, editor. *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*. Springer, 1993.
11. M. Silva. Towards a synchrony theory for P/T nets. In K. Voss et al., editors, *Concurrency and Nets*, pages 435–460. Springer, 1987.
12. M. Silva, E. Teruel, and J. M. Colom. Linear algebraic techniques for the analysis of net systems. In G. Rozenberg and W. Reisig, editors, *Advances in Petri Nets*, Lecture Notes in Computer Science. Springer, 1998. To appear.
13. E. Teruel and M. Silva. Structure theory of equal conflict systems. *Theoretical Computer Science*, 153(1-2):271–300, 1996.
14. K. Trivedi and V. G. Kulkarni. FSPNs: Fluid stochastic Petri nets. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 24–31. Springer, 1993.
15. C. Valentin and P. Ladet. Flow modelling in a class of hybrid (continuous/discrete) systems. In *IEEE Int. Conf. on Systems, Man, and Cybernetics*, Le Touquet, France, October 1993.

An Approach to the Analysis of Interworking Traders

A. Tokmakoff∗ and J. Billington‡,

∗IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
E–Mail: A.Tokmakoff@Irisa.Fr
‡ITR, University of South Australia, Mawson Lakes, 5095, Australia
E–Mail: J.Billington@UniSA.Edu.Au

Abstract. The ODP Trader provides a match–making service for Objects in an Open Distributed System. In previous work, a model of the Trader was created using Coloured Petri Nets, incorporating its capacity for standalone and co–operative Interworking amongst multiple Trader instances. In this paper, the CPN model of the Trader is analysed using Equivalence Classes (for Occurrence Graph reduction) when the Trader is configured as a standalone entity servicing multiple requests concurrently. The Trader is also analysed in a number of interworking scenarios which are, in turn, used for reasoning about complex Trader interworking topologies.

Keywords: ODP Trader; Interworking; Coloured Petri Nets; Scenario Analysis

1 Introduction

Trading [1] is an information infrastructure service which allows software entities to advertise, or *export* a service or resource to a trusted third party, known as the Trader. The service or resource being exported usually has properties associated with it, which may be static or dynamic. The Trader maintains a database of exported services which includes the type of service being advertised and its associated properties.

With the possibly widespread use of Traders, it is important that the Trading standard is error–free and unambiguous. It is also important to ensure that the Trader is as efficient as possible, especially when interworking with other Traders.

In order to engineer reliable Open Distributed Systems, there is a need for formal modelling techniques. Such techniques must be capable of capturing the essence of a system in a manner which is readable and moreover, utilise a mathematical basis that allows the system to be formally analysed. One way to ensure correctness of the Trader is to create a formal model which can be analysed to verify that the model behaves as expected under all conditions. Coloured Petri Nets (CPNs) [2] are a formalism which has been developed for modelling and analysis of concurrent systems. In this paper, we outline an approach for analysis of a CPN model of the Trader (a distributed software entity) and aim to verify its correctness.

2 The Trading Function

Trading is a fundamentally important part of the realisation of Open Object–based Distributed Systems (OOBDS) and has been the topic of standardisation by the International Organisation for Standardisation (ISO), the International Electrotechnical Commission (IEC) and the International Telecommunications Union (ITU–T) as part of their work on the Reference Model for Open Distributed Processing (RM–ODP) [3].

The RM–ODP Trading standard was developed to facilitate the dynamic location of resources or services, such as for example, a printing service in an OOBDS. It has been included as a Common

S.Donatelli, J.Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 127-146, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Object Service (COS) in the Object Management Group's (OMG) Common Object Request Broker Architecture 2.0 (CORBA) [4], a well–known Middleware Specification.

2.1 Basic Trading Entities

There are three main entities involved in the trading function, namely, the Exporter, Importer and the Trader itself. The sequence of interactions between these entities is shown in Figure 1 [6].

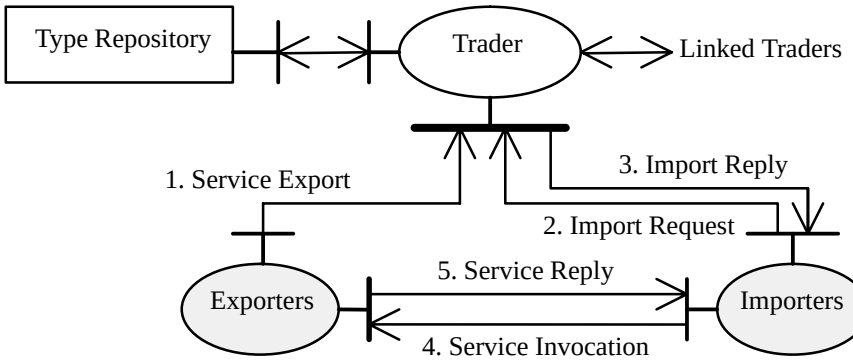


Fig. 1. Sequence of Interactions between Exporter, Trader, and Importer

1. **Service Export**: The Trader receives a service *export* from the Exporter. This request includes a description of the service, a location at which the service may be accessed, properties of the service and optional selection criteria. The service interface's type and associated properties are checked with a Type Repository to ensure legality. Details of the exported service are then stored in a database.
2. **Import Request**: The Trader receives a service *import* request (or Query) from a client. This request includes the type of service required, a list of desired attributes and optional selection criteria. The Trader checks with the Type Repository to ensure that the requested service and properties are legal.
3. **Import Reply**: The Trader searches its database of exported services and returns any successful matches after applying optional selection criteria.
4. **Service Invocation**: If a match is found, the Importer contacts the Exporter independently of the Trader to utilise the service.
5. **Service Reply**: The Exporter replies to the Importer's service request.

An example of a service that may be exported is an e–mail application, that allows software entities to e–mail data. This service could be utilised by a migrating software entity that must e–mail the results of searching activities to its owner. Since the entity does not have a fixed location, it must locate a suitable server object as it moves. By utilising the Trading service, the entity is able to locate a server by specifying the required service type and important service attributes such as for example, the ability to send binary files.

2.2 Links

The Trader uses links to other Traders for interworking. A link contains the following information [1]:

- unique Link name,
- the Lookup Interface of the Trader being linked to (used to send Queries),

- the Register Interface of the Trader being linked to (used to Register services),
- a default link follow behaviour policy,
- limiting link follow behaviour policy.

A link follow behaviour policy can take one of the following values:

- **local_only:** restricts the Query to being processed by the local Trader only. (It is not permitted to forward the Query to linked Traders),
- **if_no_local:** limits the Trader to local processing except in the case where there are no local matches to the Query. In this situation, the Trader is permitted to forward the Query to linked Traders, or
- **always:** permits the Trader to forward Queries to linked Traders.

These behaviour policies are listed from weakest to strongest and can be used to restrict the propagation of requests to linked Traders. This ordering makes it possible to calculate the smallest or minimum behaviour from a set of behaviours. This is important when multiple policies are compared as in the following examples:

```
min(always, if_no_local, local_only) = local_only
min(always, if_no_local, always) = if_no_local
```

The Trader uses `min()` and `merged_policy_options()` to *unify* its Link follow policy with those of the Query and the Link. The Lookup and Register interfaces are references to the respective interfaces of the Trader being linked to. The default follow behaviour policy is used when the importer's Query does not specify a follow behaviour. The limiting follow behaviour is the strongest follow policy that the link will allow when forwarding queries to the Trader associated with the link.

2.3 An Improved Interworking Protocol

There are a number of ambiguities and areas requiring optimisation in the Trading standard's specification of the interworking protocol [1]. The protocol ensures that infinite looping of Queries cannot occur but does not achieve this goal efficiently. In particular, the following aspects require further clarification:

- bi–directional Trading links result in wasteful Query propogation.
- the standard does not prescribe what happens when a duplicate is detected. It only states that the Query is not "processed". Does this mean that duplicates should be ignored?
- the standard does not ensure deterministic traversal of all linked Traders. This means that under certain circumstances, a Query with the same parameters may return a different list of matching Offers.

A product of the Modelling exercise was identification of these ambiguities and suggestion of alternative policies to address the issues identified. It was proposed to:

- no longer allow Queries to be propagated back to the Trader from which it came (in the case of a bi–directional link).
- return a `null` list of matching local offers when processing duplicate Queries (avoids deadlock),
- propagate duplicate Queries to *all* linked Traders when the duplicate Query has a larger `hop_count` than when it was processed before (ensures a deterministic traversal of the Offer Space).

See [7] for a more detailed discussion of the Trader interworking protocol ambiguities and proposed solutions.

3 CPN Model of the Trader

Using Design/CPN™ [5], a hierarchical model of the Trader was created containing 17 pages of CPNs (containing 109 places and 52 transitions) and 1200 lines of CPN/ML code (for defining colour sets and functions). In the CPN model, the Trader included functionality to allow Offers to be added to the Offer Space (export), Offers to be matched from the Offer Space (import) and concurrent processing of multiple Queries (multi–threading). Objects (such as the Trader, Importer and Exporter) were modelled using a hierarchy of pages and use message–passing via interfaces and the ***Comms_Medium*** shared place to communicate with each other.

The Trader Environment page (Trad_Env#2), shown in figure 2, is the *highest* page in the model. It is used to model the object instances in the system, where each instance is represented by a hierarchical substitution transition (denoted HS). The sub–page associated with each substitution transition is shown in a nearby box. When multiple instances of objects are in the system, each instance refers to an instance of the same sub–page, which has its own marking and hence, its own state.

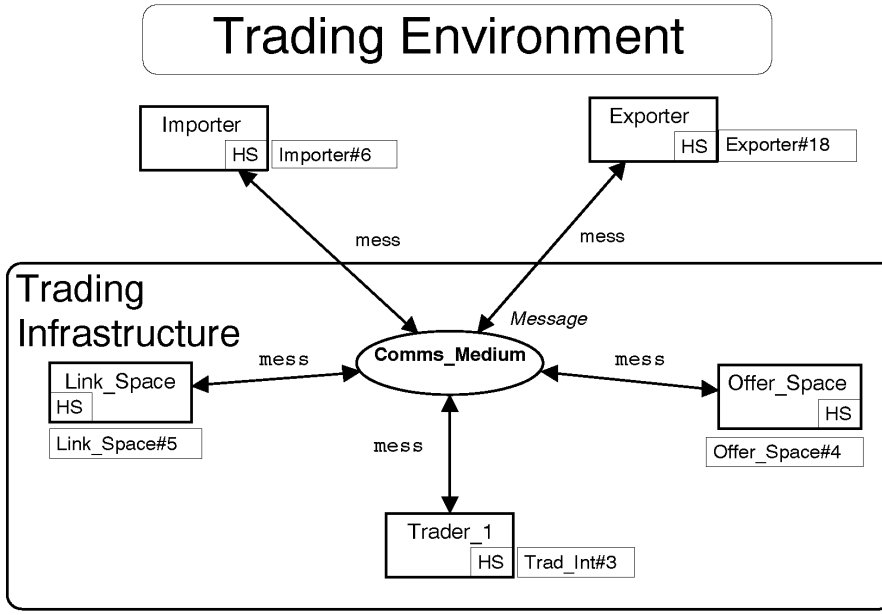


Fig. 2: Trading Environment (Trad_Env#2)

The Trader object is modelled on Trad_Int#3 (figure 3) as a group of interfaces which are connected to the ***Comms_Medium*** and also to functional operations (methods). The Trader's interfaces are grouped into three major categories: *Functional*, *Inter–Object* and *Abstract*. Functional interfaces are provided by the Trader to its clients and provide an interface to the Trading function's basic services. Inter–object interfaces are used by the Trader when communicating with the LinkSpace and OfferSpace objects (utility objects in the model). The Abstract interface provides clients with a means for inspecting Trader attributes.

Due to space limitations, a detailed presentation of the CPN model of the Trading Environment cannot be presented here. However, for more information, the reader is directed to [7,8] for a complete discussion of the model.

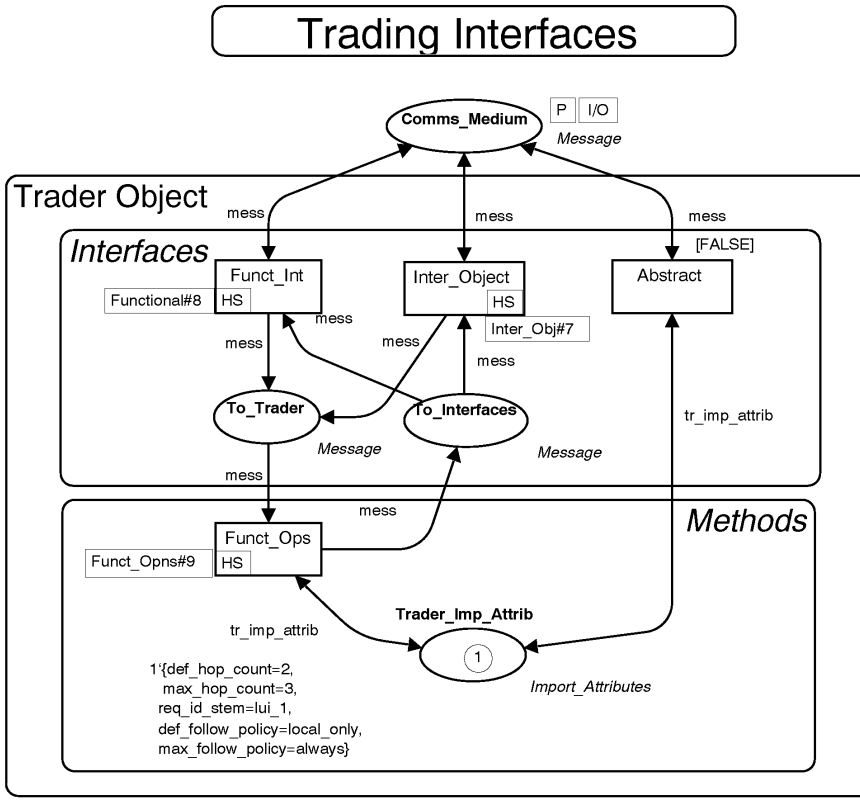


Fig. 3: Trading Interfaces (Trad_Int#3)

4 Approach to Analysis of the Trader Model

Having given a (very brief) outline of the CPN model, we now concentrate on analysing the model, with special emphasis on the interworking of Traders. Analysis of the CPN model of the Trading Environment aims to show the following:

1. correct operation of a Trader which does not interwork (***standalone***) processing a single Query.
2. correct operation of a standalone Trader with ***multiple concurrent*** Queries (internal multi–threading).
3. correct operation of ***multiple*** Traders operating concurrently and ***independently***.
4. identify and ensure correct behaviour for all ***stopping conditions*** required for Query propagation.
5. analysis of ***complex*** Trading Environment configurations.

4.1 Occurrence Graph Analysis

When the model successfully completed scenarios in the simulator, the next step was to perform Occurrence Graph (OG) Analysis. When analysing the Trader model (which possibly contained modelling errors that result in huge OGs), the technique adopted for generating OGs was a

combination of breadth–first and subsequent depth creation of the OG. By altering the OG generation branching factor via Design/CPN's `OGSet.BranchingOptions` function, it was possible to limit to 1, the number of bindings and transition elements considered when generating nodes in the graph.

In practice, this meant that for an initial period of time, the OGA tool was directed to generate the OG breadth first. This provided a large number of markings that were reached from the initial marking and could be subsequently investigated through depth–based generation of the OG. Utilising a breadth–based followed by depth–based generation of the OG made it possible to quickly narrow in on a final terminal marking which should be reachable from all nodes in the OG.

With models that are expected to terminate in the same state after all simulation runs, the OG contains a single dead node (terminal marking). This is because all of the paths that lead from the first marking of the model (node 1 in the OG) eventually reach the dead node. This node is also a home marking [2] since it can be reached by all other reachable markings. If the OG contains a single dead node and its marking is correct (based upon expected behaviour for the scenario), then the scenario has been verified.

Since the aim of the OG analysis of the model is to detect all of the dead nodes in the OG (although a single dead node is expected), this technique increases confidence in the model since dead nodes are calculated much earlier. Even though not all nodes have been calculated, the fact that all paths that *have* been calculated lead to the same dead node is a promising sign. If there is more than one dead node in the OG after using this technique, then it is safe to assume that the model is flawed. It is then possible to investigate the bindings and transitions that resulted in obtaining multiple dead nodes, which in turn leads to the cause of the modelling error.

When analysing the Trader, it became evident that modelling errors related to distinguishing between tokens within threads allowed state information of threads to become confused. This manifested itself when tokens belonging to a given thread were used by a different thread. In this situation, both threads became corrupted and the model ceased to function correctly and thus, the resulting OG contained more than one terminal marking (determined by the standard CPN/ML library function `ListDeadMarkings()`).

Simulation runs were less capable of finding these errors since, in most cases, they did not test the case where concurrent requests are at the same stage of processing within the model, the source of most errors. OG analysis of the model was much quicker and more effective at detecting concurrent thread modelling errors.

4.2 Occurrence Graphs with Equivalence Classes

To allow processing of multiple Queries concurrently (multi–threading) the CPN Trader model utilises a unique transaction identifier (`t_id`) which is associated with each Query as it is processed. When processing multiple Queries, it is possible for Queries to arrive in any order (2! possibilities with 2 Queries, and 3! with 3 Queries). This means that there are multiple paths through the OG which represent the same Trading behaviour (but use a different value for the Query transaction identifier). Having transaction identifiers allows the Trader to differentiate between tokens that are used by multiple threads of execution, but the threads operate in exactly the same way irrespective of the unique transaction identifier's value.

As demonstrated in [9], it is possible to use OGs with equivalence classes (OEOS) [10] to reduce the size of an OG. The same technique can be applied when analysing the Trader model presented in [7], where Equivalence Classes are used to map out the `t_id`s used by Traders when servicing multiple requests concurrently. This can greatly reduce the size of the OG because `t_id`s are assigned to Queries on a first–come first–served basis whilst the arrival order of Queries is non–deterministic.

Without using Equivalence Classes, the resulting OG includes nodes representing use of different `t_id`s for each Query, which enlarges the size of the OG without providing extra analytical benefit (since the value of a Query's `t_id` must be unique within the Trader, but its value is unimportant). Thus, Equivalence Classes can be used when generating an OG to detect markings that are equivalent and to represent them using a single node in the OG, thereby reducing its size.

Projection functions (`EquivBE()` and `EquivMark()`) which map out the `t_id` from tokens (such as those of colour set Message and Op_Data) were created. Using these functions, the OEOS tool was used to calculate reduced Equivalent OGs for scenarios that included concurrent Query servicing by a Trader. However, the projection functions were not created to map out Trader object identifiers.

5 Analysis of a Trader Configured as a Standalone

A fundamental step in the analysis of interworking Traders is the analysis of a Trader which is configured to be a standalone Trader. This means that the Trader does not interact with other Traders when processing a Query. The analysis must include all permutations of relevant initial conditions and the servicing of multiple requests by concurrently executing threads of control. In order to verify the correct operation of the Trader's Query operation as a standalone entity, the model must include at least one instance of the following objects: Trader, Importer, OfferSpace and LinkSpace. For analysis of the Export operation, an Exporter must also be included in the model.

5.1 Single Non–threaded Trader Query

A single non–threaded Trader is the simplest of scenarios in which the Trader has a single Query to service and does forward the Query to linked Traders. This scenario is shown in figure 4, where an Importer is represented by a dashed circle containing an **I**, and the Trader is represented by a circle containing a **T** and an instance identifier (**1** or **2**). When analysing a standalone Trader, it is possible that the Query is not propagated to linked Traders due to policy limitations rather than a lack of links to follow. This is shown in figure 4, where a dashed link from **T1** to **T2** exists but is not pursued by the Query (shown as a dashed arrow). Note that verfication of theTrader's Export functionality is presented in [7] but is omitted here due to space limitations.

OG analysis of the Trader can be used to verify that the Trader successfully performs a Query operation and always finishes the Query with exactly the same marking. This marking should indicate that:

- a response to the Query containing matching offers associated with all Traders visited by the Query was received by the Importer,
- all objects are in a *clean* state (no unexpected tokens remain in an object's CPN).

All possible permutations of the stopping conditions need to be considered when verifying the scenario in figure 4. To ensure that the model behaves correctly under all conditions, each of the possible return values for `merge_policy_options()` must be used with:

- `hop_count=0` and `hop_count<>0` and
- `links available=0` and `links_available<>0`

Table 1 shows the results of OG verification for each of the scenario permutations when the number of available links`=0` (i.e. the LinkSpace object returns an empty list of links). Two unique behaviours were identified in the scenarios, resulting in two classes of OG (A,B).

Table 1 shows that the model displays identical behaviour when the unified policy`=local_only` and when the `hop_count=0`. This behaviour is denoted as type A. For combinations which resulted in a type A OG, the Trader did not issue a request to the LinkSpace object to `get_links` (since the Query does not need to be propagated), resulting in a small OG (20 nodes).

In contrast, combinations resulting in type B OGs did contain a reference to the LinkSpace object to `get_links`. This is expected since when `merged_policy_options()`= `if_no_local` or `always`, the Trader needs to get links from the LinkSpace object, regardless of whether the links are subsequently used or not. This depends upon the results obtained from a local search of the OfferSpace and the link's follow policy.

Table 1 – Combinations of hop_count and follow_option for links=0

Available links=0		merged follow_option			
		local_only	if_no_local and local_match=true	if_no_local and local_match=false	always
Query hop_count	= 0	A	A	A	A
	<> 0	A	B	B	B

```
A: Nodes=20, Arcs=21 Full OG. Simulation = 20 steps
B: Nodes=75, Arcs=122 Full OG. Simulation = 26 steps
```

It follows that combinations resulting in type A OGs are not affected by changing the number of links associated with a Trader because the links are never obtained. Thus, only combinations resulting in type B OGs need to be investigated further when the number of links available is non–zero, as shown in Table 2.

Table 2 – Combinations of hop_count and follow_option for links<>0

Available links<>0		merged follow_option			
		local_only	if_no_local and local_match=true	if_no_local and local_match=false	always
Query hop_count	= 0	N/A	N/A	N/A	N/A
	<> 0	N/A	C	**link search**	**link search**

```
C: Nodes=85, Arcs=140 Full OG. Simulation = 28 steps
```

When the `merged_policy_options()`=`if_no_local`, the `hop_count<>0`, a local match to the Query is found and there are links to follow, the Trader exhibits a third behaviour denoted C. This behaviour is observed because the Trader obtains links from the LinkSpace object and performs a local offer space search which results in a matching offer, thereby signalling that the search is not to be propagated further. Some combinations with non–zero links result in the Query being propagated along a link (marked as **link search**) which is beyond the scope of analysis for a standalone Trader (see section 7).

The Trader processes `if_no_local` requests with an extra two transition occurrences when compared with type B behaviour. The Trader aborts searching linked Traders when it determines that the local search resulted in a successful match. Since behaviour C exhibits two more transition occurrences than behaviour B, it has a larger OG.

By using CPN/ML's `ListDeadMarkings()` function call, it is possible to verify for each OG that it contains a single terminal marking. This means that the model reaches a single conclusive state whose marking may be checked to ensure that it is the expected terminal state of the model.

At this point, the Trader's behaviour has been verified for all possible permutations of its controlling parameters and this result can now be used as a building block for further verification.

5.2 Single Trader with Concurrent Request Servicing

There is a need to verify that multiple concurrently executing threads within the Trader do not interact in an undesired manner. Since there is a high degree of internal concurrency within the Trader itself, applying Occurrence Graphs with Equivalences reduces the size of the model's OG by mapping out the `t_id` field inside tokens. This field is used by the model to distinguish tokens belonging to different threads. The identifier associated with a thread depends upon the order in which it is processed. This is a non–deterministic property of the model and is mapped out by the Equivalence relation used in the Design/CPN OEOS tool [5].

In figure 5, the Trader is sent two Queries (Query 1 and Query 2) by two Importers (**I1** and **I2**). These requests are generally for different services, although this is not critical for the verification of independent concurrent service processing. It is also possible that the two Queries originate from the same Importer (e.g. **I1**), rather than from two Importers.

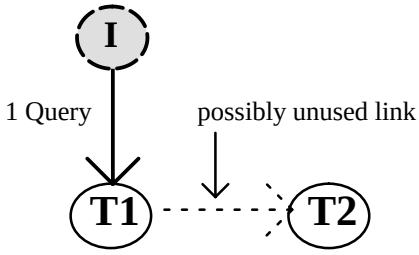


Fig. 4. Trader servicing a single Query

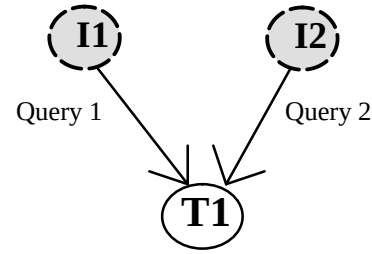


Fig. 5. The Trader services multiple Queries concurrently

In section 5.1, three fundamental standalone Trader behaviours were identified (A, B and C). When verifying that the Trader can service multiple Queries concurrently, it is important to test all possible combinations of Trader behaviours. Having identified the three fundamental Trading behaviours, only combinations of behaviour A,B and C must be investigated. The following combinations have been identified:

```
Query1=A, Query2=A or Query1=A, Query2=B or Query1=A, Query2=C or
Query1 = B, Query2 = B or Query1=C, Query2=C
```

Thus, when a Trader services a type A Query, it displays behaviour of a type A OG. Similarly, servicing a type C Query will result in a type C OG. Table 3 shows the number of steps required for simulation runs using different combinations of concurrent Queries that are processed by a standalone Trader. Due to symmetry, some of the scenarios are redundant since they are already included in the table.

In order to display type B behaviour, the Trader must not have any links available, whereas type C behaviour requires the Trader to have at least one link available in the LinkSpace object. Thus, it is not possible for a Trader to service a type B Query at the same time as a type C Query since they are mutually exclusive (Mut–Ex).

The simulation step values in Table 3 are as expected, based upon the number of steps required to simulate single type A, B and C behaviours (calculated in section 5.1). e.g.,

```
A concurrent A = 20+20 = 40,  A concurrent C = 20+28 = 48
                B concurrent B = 26+26 = 52
```

Table 3 – OG/OEOS size and simulation steps for Single Trader Concurrent Queries

Concurrent Behaviours		Query 1		
		A	B	C
Query 2	A	Nodes=680 **(399)** Arcs=1284 **(773)** Secs=11 **(5)** 40 steps	Redundant	Redundant
	B	Nodes=2630 **(1547)** Arcs=6756 **(3954)** Secs=74 **(26)** 46 steps	Nodes=10124 **(4274)** Arcs=32820 **(13163)** Secs=507 **(97)** 52 steps	Mut–Ex
	C	Nodes=2992 **(1753)** Arcs=7740 **(4516)** Secs=89 **(29)** 48 steps	Mut–Ex	Nodes=13104 **(4096)** Arcs=42956 **(12323)** Secs=653 **(91)** 56 steps

In addition, table 3 also provides statistics for standard OGs (no eqiuvalance classes used) and OGs using Equivalance Classes (in **bold**) for each combination of concurrent Query types. For each OG described in Table 3, there is a single terminal marking which corresponds to successful termination of the concurrent Queries.

The result for two concurrent A Queries is expected since for each in Query A there are 20 different possible states for Query A', resulting in 400 different states. The terminal marking of this example combines two of the states into one, resulting in 399 states using Equivalence Classes. OEOS analysis resulted in significantly smaller OGs whilst still detecting the single terminal marking in each case.

5.3 Summary

In this section, the Trader's most basic functionality has been verified, where a single Query and concurrent Queries are processed by a standalone Trader. All of the possible stopping criteria were used in the verification and it was found in section 5.1 that the Trader's behaviour was identical when `hop_count=0` and `merge_policy_option()=local_only` (type A OG).

The three unique behaviour types may be summarised as:

A: The Trader does not attempt to get its links,

B: The Trader gets its links but finds that there are no links to pursue,

C: The Trader has a unified policy of `if_no_local`, obtains valid links from the Link-Space object but does not send the Query to linked Traders because it also obtains a local OfferSpace match to the Query.

In section 5.2, a single Trader was configured in scenarios to concurrently service each of the basic behaviour pairs. This showed that the Trader is capable of servicing all types of Queries concurrently without exhibiting undesirable behaviour (deadlock). The results presented in this section will now be used for further analysis of multiple independent Traders in following sections.

6 Analysis of Multiple Independent Traders

In section 5, the basic standalone Trader functionality was verified for all permutations of parameters (policies, links and offers), thereby testing all standalone Trader stopping conditions for single and

concurrent Query processing. In this section, the previous result is used to verify the correct operation of multiple Traders operating independently but concurrently in the same system.

6.1 Multiple Autonomous Objects

It is important to verify that the model can contain multiple autonomous entities which do not "interfere" with each other. An abstract model of concurrent independent Traders is shown in figure 6, where Traders (**T1** and **T2**) service 2 Queries independently.

Since there is no internal thread concurrency within the Traders, there is no benefit to be obtained by using OEOS to generate the OG. Using the OGA, it was found that there were no deadlocks since a single dead node was observed in the OG (table 4). The single terminal marking in the OG indicates that for all possible behaviours, the model always terminates in the same desirable state.

In section 5.1, it was shown that when issued as the sole Query to a standalone Trader, OGs for servicing of Type A, B and C Queries all contained a single terminal marking. In this section, it has been shown that combinations of Query types can be serviced concurrently by two independent Traders.

Thus, it has been shown that the model is capable of allowing multiple instances of Traders to operate independently and concurrently. This result may appear intuitive, but it ensures that the CPN model of the Trader does not allow any hidden interactions between Traders when there are multiple Trader instances operating concurrently.

Table 4 – OG size for Multiple Traders

Concurrent Behaviours		Trader 1 Query		
		A	B	C
Trader 2 Query	A	Nodes=437 Arcs=828 Secs=2	Redundant	Redundant
	B	Nodes=1547 Arcs=3954 Secs=37	Nodes=5429 Arcs=17376 Secs=142	Mut–Ex
	C	Nodes=1753 Arcs=4516 Secs=43	Mut–Ex	Nodes=6969 Arcs=22584 Secs=161

6.2 Multiple Autonomous Threaded Objects

The final step in the basic verification of the Trader is to verify that the model can accommodate multiple Trader instances that are autonomous, operate concurrently and independently and can service multiple requests concurrently. This will be accomplished using the results from previous sections.

In section 5.2, it was verified that the Trader model was capable of successfully servicing multiple Queries concurrently. Section 6.1 showed that the model is capable of containing multiple independent Traders. Using these two results, it can be deduced that the model allows multiple independent Traders to operate concurrently (from 6.1), where each Trader is able to service multiple Queries concurrently. This is shown in figure 7.

This result is significant because it allows complicated Trading scenarios to be broken down into more manageable independent concurrent activities which can be verified individually. Since it has been

shown that multiple concurrent Traders do not interact in unexpected ways, it is possible to break down complicated scenarios into a number of simple scenarios which operate concurrently but independently. This result will be used when verifying more complex interworking Trader scenarios in section 7.5.

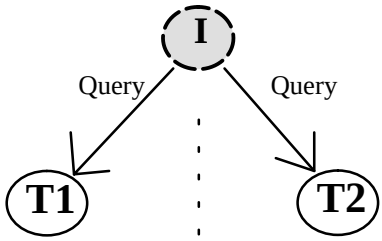


Fig. 6. 2 Traders and 2 independent Queries

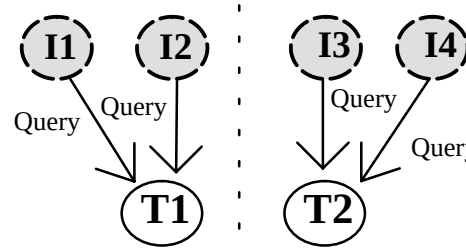


Fig. 7. 2 Traders and 4 independent Queries

7 Analysis of Interworking Traders

To verify the interworking of Traders, it is necessary to create a series of scenarios in which multiple traders interact. These scenarios will verify a set of Query stopping conditions which can be used to verify that forwarded Queries always terminate. The analysis also ensures that the model does not contain livelocks or deadlocks. The analysis builds upon the results of the previous sections and aims to verify the interworking of `m` Traders, each of which is capable of servicing `n` requests concurrently, where `n` and `m` are positive integers.

7.1 Recursive Definition of Interworking Traders

When a Query is propagated to linked Traders, the forwarded Query is a slightly modified version of the initial Query since it has a smaller `hop_count` and possibly, a different `follow_policy`. A propagated Query is serviced by a different Trader using the same type of interface, but the Query has slightly different parameters to those of the initial Query. This resembles a recursive situation in which a function calls itself many times with different parameters until it reaches a point at which it no longer calls itself. At that point, it retraces its path through the recursive function calls until it reaches the initial call and returns its result.

According to Garland [11], a recursive problem must meet two requirements if it is to solve a problem successfully. They are:

- any invocation of a recursive sub–program from within its own definition must solve a problem simpler than the one used when the program was first invoked,
- there must be some instances in which the recursive sub–program does not invoke itself recursively (stopping conditions).

A recursive approach to algorithms is common–place in functional programming languages such as SML, because it allows a small function to re–call itself until the problem has been solved. It is considered that Traders do the same thing when they interwork, since a Query is continually broken up into smaller, more easily processed elements which are forwarded to linked Traders. Each such Query should eventually reach a point at which it is no longer propagated and the Query trail is re–traced back to the Trader which received the initial Query.

The analysis of the Trader in this section exploits the recursive behaviour displayed by the Trader and combines it with a number of stopping conditions to verify that forwarded Queries always terminate. In addition, it will be shown that when an offer is propagated between Traders, its path through the Trading graph is deterministic.

7.2 Reducing the size of the Problem

The Trader reduces the size of the problem (forwarding of Queries to linked Traders) in two ways. Firstly, it reduces by `1` the `hop_count` in a Query's Import Policy when it is forwarded to linked Traders. It is also possible for the `follow_policy` of a Query to be reduced in strength as it is propagated, either by the Trader itself, or the link which the Query is following (i.e. from `if_no_local` to `local_only`). Using either of these two parameters, it is possible to limit the number of links the Query follows, thereby reducing the size of the problem.

7.3 Interworking Query Stopping Conditions

When modelling the Trader, the following six Query stopping conditions were identified. Each of the conditions have been modelled in an associated scenario as shown in Table 5. A *valid* link is one that is not a reverse link back to the Trader from which the Query was received [7].

Table 5 – Scenario Stopping conditions

Scenario	hop_count=0	total available links = 0	merged policy = local_only	merged policy = if_no_local but gets a local match	total available links<>0 but valid links = 0	Query re–visits a Trader with a smaller hop_count
1	Y	N	N	N	N	N
2	N	Y	N	N	N	N
3	N	N	Y	N	N	N
4	N	N	N	Y	N	N
5	N	N	N	N	Y	N
6	N	N	N	N	N	Y
Combined	Y	Y	Y	Y	Y	Y

The scenarios are used to verify that a Query will stop being propagated when a specific stopping condition is met. The scenarios have been designed to ensure that only one of the stopping conditions is tested in each scenario except for the combined scenario which includes **all** of the stopping conditions (although some stopping conditions are tested more than once). Analysis of these stopping conditions will be used later in section 7.5 to analyse a more complex Trading example.

7.3.1 hop_count=0 (Scenario 1)

This scenario verifies that a Query stops following links and returns a result (possibly an empty list of Offers) when its `hop_count=0`. This is a very simple stopping condition which is shown in figure 8.

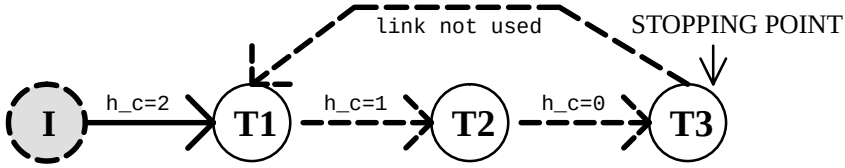


Fig. 8. Query stops when `hop_count=0`

The Query is sent by the Importer (**I**) with `hop_count=2` and is processed by Trader 1 (**T1**) which also forwards the Query to Trader 2 (**T2**) with `hop_count=1`. It processes the Query and forwards it to Trader 3 (**T3**) with `hop_count=0`. **T3** cannot forward the Query (since `hop_count=0`) and thus, it processes the Query locally and returns matching offers to **T2** and so on until the combined result of all the searching is returned to **I**.

The scenario verifies that even with a merged policy option of `always` and *valid* links to follow such as the link from **T3** to **T1**, (see section 7.3.5 for an example of a non–valid link), the Query terminates after being serviced by **T3**. Table 6 contains statistics related to all the OG of scenarios presented in this paper. The marking of the single terminal node shows that the Query is propagated to **T3** but is not propagated onwards to **T1** since **T1** serviced the Query when its `hop_count=2`.

Table 6 – Results of OG Analysis for Scenarios

Scenario	1	2	3	4	5	6	Combined
Nodes	2628	758	296	844	758	16005	5848
Arcs	6807	1789	535	2015	1789	54337	7485
Seconds	25	6	2	7	6	182	512
Status	FULL	FULL	FULL	FULL	FULL	FULL	PARTIAL

7.3.2 Total available links=0 (Scenario 2)

In this scenario the total number of links available to the Trader for forwarding of Queries is zero, as shown in the scenario model of figure 9. The two Traders are configured such that the `limiting_follow_rule=always` in all cases, and the Query `hop_count` starts with a non–zero value (arbitrarily chosen to be 2). However, the LinkSpace object is initialised so that it returns an empty list of links for **T2** to follow. Thus, even though the Query `hop_count` allows more hops to be made, the Query terminates after being processed by both **T1** and **T2**.

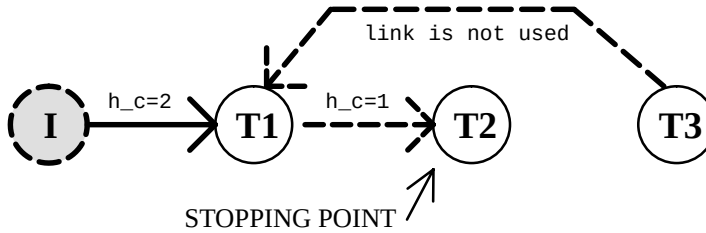


Fig. 9. Query stops when there are no links to follow

7.3.3 Unified Policy follow behaviour=local_only (Scenario 3)

In this scenario (shown in figure 10), the Query stops after being processed by Trader 1 (**T2**) even though there are valid links to be followed and the Query's `hop_count` is non–zero. The model performs a local search of the OfferSpace object since the request is unique, but **T2**'s links are not searched since `merge_policy_options()` evaluates to `local_only`.

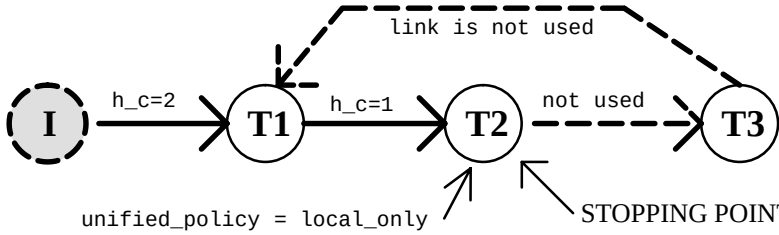


Fig. 10. Query stops when `unified_policy=local_only`

As with previous scenarios, the OG graph (see table 6) contained a single terminal marking which indicated that a matching offer from **T2** was returned to **I**. It also showed that the Query was not

forwarded by **T2** to **T3** even though `hop_count=1`, and a legal link to **T3** could have been followed. Instead, due to **T2**'s `max_follow_policy` value (`local_only`), the `merged_policy_option()` equalled `local_only`, thereby terminating propagation of the Query to linked Traders at **T2**.

7.3.4 Merged Policy=if_no_local, Local OfferSpace Search Is Successful (Scenario 4)

The scenario shown in Figure 11 aims to verify that a Query stops being propagated at **T2** when the Trader has a merged policy of `if_no_local` and a local match is obtained, even though there are legal links to pursue (from **T2** to **T3**) and the Query's `hop_count` is non–zero. This scenario is similar to the scenario described in section 7.3.3 except that **T2**'s `max_follow_policy=if_no_local` and thus, `merge_policy_options()` returns `if_no_local` rather than `local_only`.

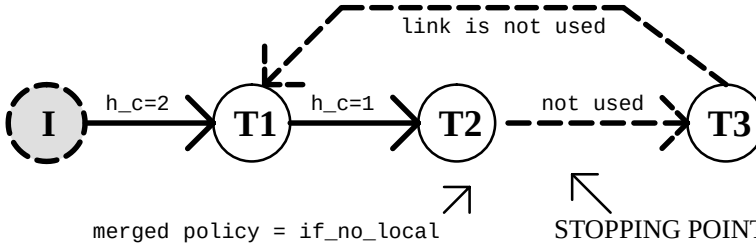


Fig. 11. Query stops with a local match and merged `follow_policy=if_no_local`

An automatic simulation run of this scenario took 59 steps with the `hop_count` of the Query initialised to 2. OG analysis of the scenario is shown in table 6. A single terminal marking was obtained which indicated successful termination of the Query.

7.3.5 Valid links to follow=0 (Scenario 5)

This scenario (shown in figure 12) illustrates one of the modifications to the Trading standard's interworking protocol [1] outlined in section 2.3. It is a specialised case of when there are no links to follow since the only link that exists is not considered *valid* according to the modified interworking protocol since it is a bi–directional link between Traders.

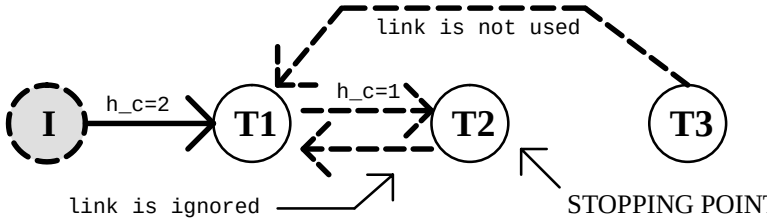


Fig. 12. Query stops when there are no valid links to follow

In the scenario, the Query has a `hop_count=2` and according to the unmodified Trading standard [1], the Query should be propagated by **T2** back to **T1**. This is a redundant forwarding since the Query has already been processed by **T1** with a greater value of the `hop_count` parameter than the Query which would be re–sent to it. There are no situations in which a greater Trading scope will be achieved, assuming a static Trading environment (links do not change often over time). An automatic simulation of this scenario required 57 steps.

This protocol optimisation results in a significant reduction in the wasted processing by Traders and messaging bandwidth when interworking, since unproductive Trader queries and messaging is reduced.

7.3.6 Revisit Trader with Smaller hop_count (Scenario 6)

In the scenario shown in Figure 13, a Query request is initiated by the Importer with a `hop_count=4`, and all policies are such that the Query is propagated successfully to all links. Since **T3** is connected via a link to **T1** when the Query's `hop_count=1`, the Query could possibly continue to be re–processed by **T1**, and subsequently passed on to **T2** at which point the `hop_count` would reach 0. Both of these propagation actions are redundant since the Query operation has already been processed by **T1** and **T2**. Therefore, propagation of the Query is terminated at **T1** because it is a duplicate Query operation whose `hop_count` is less than the previously processed Query operation.

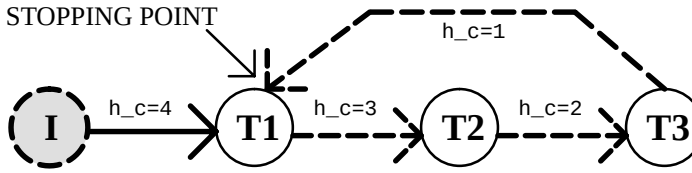


Fig. 13. Query stops when it re–visits a Trader

7.4 Deterministic Traversal of the Offer Space

As discussed earlier, the Trader's interworking protocol as specified in the Standard is not deterministic. In this scenario (shown in figure 14), **T1** receives the same Query from two sources, where each Query has a different value for `hop_count`. The order in which the Queries are processed is not deterministic and thus, either of the two Queries may be serviced first. Without also maintaining the Query's `hop_count` parameter in the history of *recent* interworking Queries, it is not possible for the Query from **T4** to deterministically reach **T2**. This policy results in two terminal markings which correspond to:

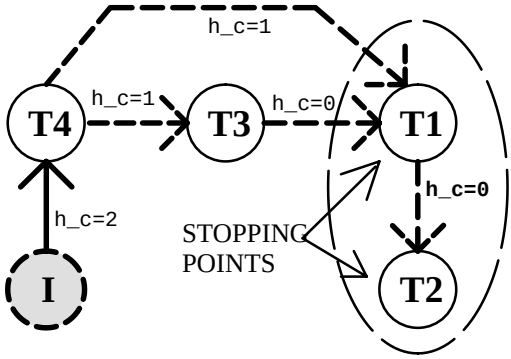
- when the **T3** Query is serviced first and the **T4** Query is rejected as being a duplicate, thereby returning an empty list of matching Offers, and
- when the **T4** Query is serviced first and manages to propagate to **T2** where it obtains a matching Offer.

Figure 15 simulates the scenario of figure 14 and focuses on **T1** and **T2**'s behaviour, as indicated by the dashed ellipse in figure 14. Using initial markings, it is possible to simulate the effect of the Query having been serviced and propagated by **T4** and **T3**. This reduces the size of the OG since it is not necessary to include the Query servicing of **T4** and **T3** in the analysis.

A simulation of the scenario in figure 15 required 67 steps. An OG for the modified protocol which maintains the Query `hop_count` was calculated and its statistics are shown in Table 6. The single terminal marking showed an empty list of Offers being returned to **T3** and a list containing a single matching Offer being returned to **T4**. This was expected since the OfferSpace object was initialised so that **T2** would obtain a matching offer to the Query whereas **T1** would not.

7.5 Combined Trading Topology (All Stopping Conditions)

In this scenario, all of the stopping conditions outlined in section 7.3 are demonstrated using a single scenario. The size of this scenario's OG would be prohibitively large due to the high degree of inter–object concurrency and large number of active Traders. In Figure 16, the termination of a Query as a result of each of the six stopping conditions is indicated by an arrow and a corresponding number.



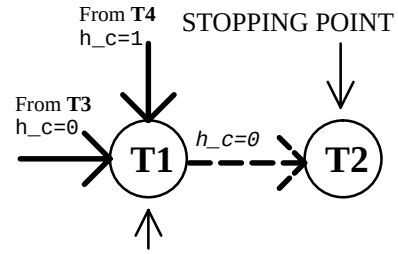


Fig. 15. Non–deterministic Query propagation

Fig. 14. T1 gets duplicates with different hop_count values

When a Query from the Importer is processed by **T1**, it is forwarded to both **T2** and **T5**. Each of these Queries continue to propagate until all (sub)propagated Queries enter one of the stopping conditions indicated in section 7.3. The path that a Query takes through the linked Traders will be denoted a *trace* and each of these conditions (which are represented at least once in the Combined Scenario) are discussed below.

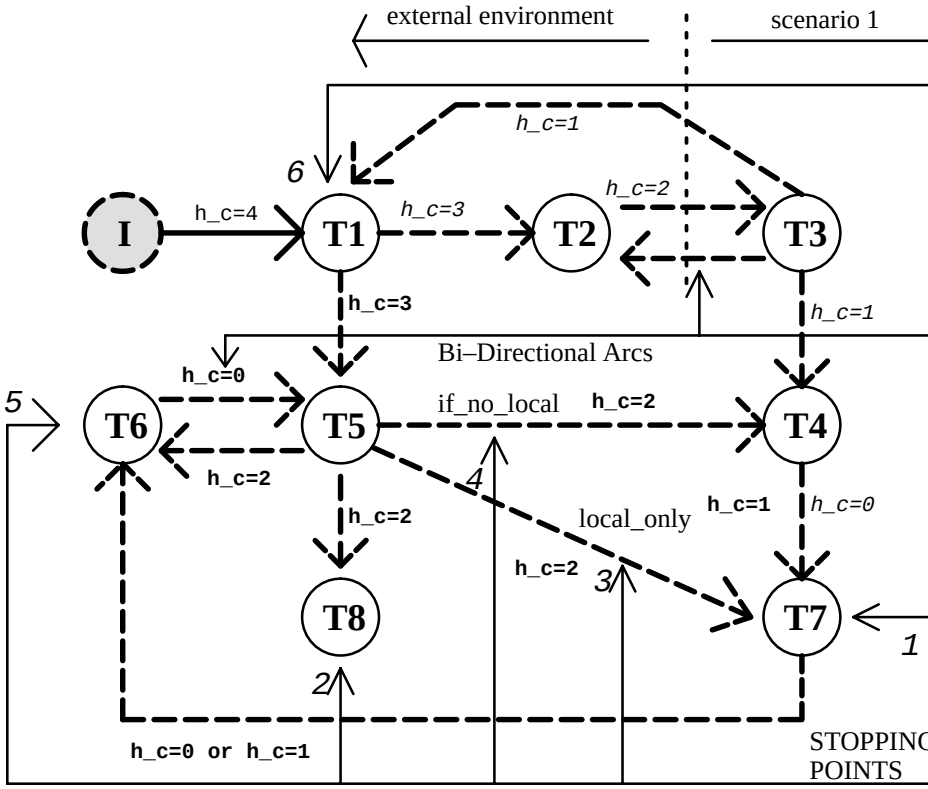


Fig. 16. Combined Scenario

7.5.1 Stopping Condition 1

The trace shown in figure 16 (terminating at *1*) shows a Query that propagates from the Importer **I** through **T1**, **T2**, **T3**, **T4** and then stops at **T7** because its `hop_count=0`. This is comparable to the stopping condition explained in section 7.3.1 except that in this example, the Query visits 5 Traders rather than 3. The propagation of Queries is linear and it has been shown that Traders can successfully propagate Queries for 3 links. Consider a partitioning of the Query in figure 16, scenario *1* into two elements: the external environment and scenario 1.

The external environment must:

- decrement `hop_counts` by 1 and
- accept results from linked Traders and return them to the invoking entity.

This behaviour has already been demonstrated in the scenario of section 7.3.1 between **T2**, **T3** and **T4**. Thus, the behaviour of the external environment is equivalent to a sub–set of a previously verified behaviour scenario. The trace for scenario *1* presented in figure 16 can be projected onto the scenario analysed in section 7.3.1, where **T2**=**I**, **T3**=**T1**, **T4**=**T2** and **T7**=**T3**. Thus, this trace can be considered equivalent to the serial concatenation of two analysed scenarios, both of which are guaranteed to be deadlock–free and correct.

7.5.2 Stopping Condition 2. This stopping condition is is the same as that analysed in section 7.3.2 except that the Query terminates (at *2*) with `hop_count=2`, rather than with `hop_count=1`. The Query terminates because it runs out of links to follow which is independent of `hop_count`'s value (as discussed in section 7.3.2).

7.5.3 Stopping Condition 3. This stopping condition is similar to that analysed in section 7.3.3, except that the Query terminates (at *3*) with `hop_count=3` rather than `hop_count=1`. In either case, the `hop_count` value is non–zero, as discussed in section 7.3.3. A mapping (from scenario to trace) can be used to compare the scenario of section 7.3.3 with the trace of figure 16. **I**=**I**, **T1**=**T1**, **T2**=**T5**, **T3**=**T7**.

7.5.4 Stopping Condition 4. This stopping condition is similar to that analysed in section 7.3.4, except that the Query terminates (at *4*) with `hop_count=3` rather than `hop_count=1`. (Termination is independent of `hop_count` – see section 7.3.4).

7.5.5 Stopping Condition 5. This stopping condition is similar to that analysed in section 7.3.5 except that the Query terminates (at *5*) with `hop_count=2` rather than with `hop_count=1`. Termination of the Query in this scenario is independent of `hop_count`, as discussed in section 7.3.5.

7.5.6 Stopping Condition 6. This stopping condition is identical to that analysed in section 7.3.6 which has been shown to behave correctly.

7.6 Discussion

By applying the recursive Trading approach of section 7.1, it is possible to state that this complicated scenario is deadlock–free and is guaranteed to terminate with a single terminal marking if its OG was generated. This is because each of the linked Queries that are created as the initial Query propagates

from **I** to **T1** and onwards have been verified individually in preceding sections. The fact that multiple concurrent Trader processing has also been shown to be **independent** (section 6.2) allows us to decompose the scenario into multiple concurrent smaller scenarios which have already been verified. This means that complicated scenarios such as that shown in figure 16 can be analysed as a series of concurrent traces which do not interact and can be analysed individually. This avoidins the state space explosion problem associated with the OG for multiple concurrent Query propagation. Figure 17 shows Query propagation with visited Traders shown as grey ellipses. If using only `hop_count=0` or `available_links=0` as stopping criteria, all nodes in the Query propagation graph of figure 17 would be visited. However, by utilising more stopping criteria, the larger, finite graph is "pruned" to only those Traders shown in grey.

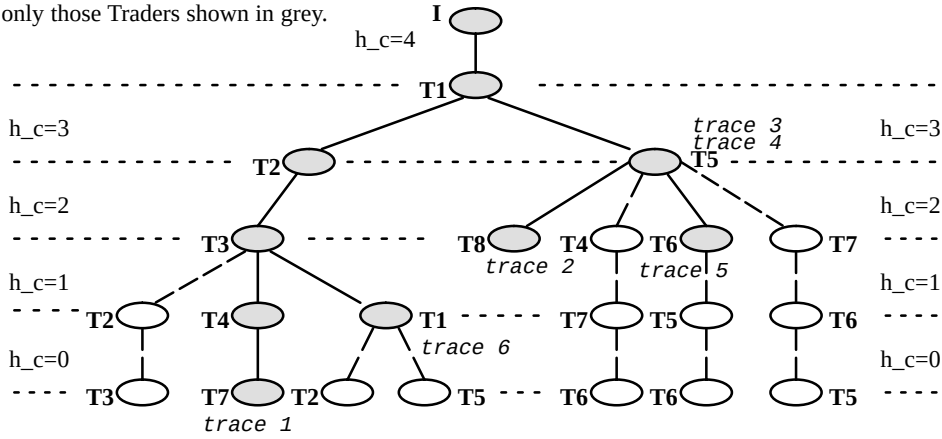


Fig. 17. Concurrent independent traces using hop_count=0 as a stopping condition

7.7 OG Analysis of Combined Scenario

The CPN model of the Trading Environment was configured as depicted in the scenario diagram of Figure 16. Although the creation of a complete OG for the scenario was not possible, analysis was performed using simulation and the creation of a partial OG. The simulation run resulted in a list of Offers being returned to the Importer as expected and took 303 steps to complete. The partial OG was created by allowing the OG to be created normally (breadth–wise) for 10 seconds and then forcing it to create the OG depth–wise until complete. This resulted in a partial OG, which did not include all possible scenario behaviours but did narrow in on a single terminal marking (593) as expected.

This process was repeated a number of times and the same single terminal node (593) was evident. This is a good result since it improves confidence in the assertion that the combined scenario has been verified using the previous scenario verification results.

8 Conclusions

In this paper, an approach to the analysis of a large CPN model was presented. The approach utilises results from a number of small verified scenarios to build up a view of more complex Trader interworking scenarios. The analysis was performed firstly using simulation and subsequently utilising OGs and OGs with Equivalence classes. The aim of the analysis was to test the correctness of the model and therefore the computational viewpoint of the Trading standard.

Correct operation of the Trader was verified when servicing single and multiple Queries concurrently as a standalone entity. It was also shown that the model behaves correctly when there are multiple instances of objects which operate concurrently and independently.

Using these results as a basis, it was shown that a more complex Trader connection topology could be analysed by decomposing it into a number of parallel independent Query propagation scenarios which were verified in this paper. Further work remains regarding formalisation of this approach to reasoning about arbitrarily complex interworking topologies.

Acknowledgements

The authors would like to thank Philippe Darondeau and the anonymous reviewers for useful comments regarding the draft of this paper.

References

[1] ISO/IEC Trading (1997) "*Reference Model of Open Distributed Processing – Trading Function*", ISO/IEC IS 13235–1. ITU/T Draft Rec. X950–1, 1997. Also available at: **http://www.dstc.edu.au/AU/research_news/odp/trader/standards.html**

[2] Jensen, K. (1992) "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1 : Basic Concepts"*, EATCS Monographs on Theoretical Computer Science, Springer–Verlag.

[3] ISO/IEC IS 10746–1 | ITU–T Recommendation X.901 (1996) "*Reference Model of Open Distributed Processing – Part 1: Overview and Guide to Use"*, Geneva, Switzerland.

[4] The Object Management Group Inc (1995), *"The Common Object Request Broker: Architecture and Specification Revision 2.0"*, July, 1995.

[5] CPN Group, University of Aarhus (1996), Online Design/CPN User's Manual. **http://www.daimi.aau.dk/designCPN/man/X_2.0/REF/Reference.All.pdf**

[6] ISO/IEC Trading Tutorial (1996) "*Reference Model of Open Distributed Processing – Trading Function Annex A: Tutorial of the Draft Trading Function*". **http://www.dstc.edu.au/AU/research_news/odp/trader/tr_tutorial.html**

[7] Tokmakoff, A. (1998) "*Modelling, Analysis and Prototyping of the ODP Trader using Coloured Petri Nets and Java"*, PhD Thesis, Faculty of Information Technology, University of South Australia, March 1998. **http://www.itr.unisa.edu.au/~steven/thesis/aat.pdf**

[8] Tokmakoff, A. and Billington, J. (1996) *"CPN Modelling of An Object Based System: The ODP Trader"*, 1st International Workshop on Formal Methods for Open Object–based Distributed Systems, Paris, France, May 1996, pp. 245–260. **http://www.itr.unisa.edu.au/people/andrew/papers/FMOODS96.ps.gz**

[9] Tokmakoff, A., Billington, J. (1998) *"Reachability Analysis of the ODP Trader using Equivalence Classes"*, Proc. International Conference on Software Engineering: Education and Practice, Dunedin, New Zealand, 26–29 January, 1998, IEEE Computer Society Press. **http://www.irisa.fr/prive/Andrew.Tokmakoff/tsec/papers/SEEP98.pdf**

[10] Jensen, K. (1994) "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2 : Analysis Methods"*, EATCS Monographs on Theoretical Computer Science, Springer–Verlag.

[11] Garland, S., J., (1986), "*Introduction to Computer Science with applications in Pascal*", Addison–Wesley, 1986, ISBN 0–201–04398–X.

Parallel Approaches to the Numerical Transient Analysis of Stochastic Reward Nets

Susann Allmaier and David Kreische

Lehrstuhl für Rechnerstrukturen (IMMD III), Universität Erlangen-Nürnberg,
Martensstr. 3, D-91058 Erlangen, Germany
{snallmai|ddkreisc}@informatik.uni-erlangen.de
http://www3.informatik.uni-erlangen.de:1200/Projects/panda.html

Abstract. This paper presents parallel approaches to the complete transient numerical analysis of stochastic reward nets (SRNs) for both shared and distributed-memory machines. Parallelization concepts and implementation issues are discussed for the three main analysis steps that are (1) generation of the underlying continuous-time Markov chain (CTMC), (2) solving the CTMC numerically for the desired time points and (3) converting the results back to the net level by evaluating reward based result measure functions. The distributed-memory approach implements dynamic load balancing mechanisms in step (1) to guarantee an equal distribution of the state space onto the main memories of the clustered machines. The shared-memory algorithms are based on elaborated synchronization mechanisms which allow parallel read and write access to the global irregular data structure of the CTMC. Performance measurements on different architectures and a comparison of the approaches are given. All the algorithms are integrated in ℙPANDA which consequently is a parallel SRN modeling tool suitable for different multiprocessor platforms.

1 Introduction

Background. One of the most pressing problems for the numerical analysis of GSPNs [5,16] is the state-space explosion which is therefore the topic of many papers in this field. The proposed approaches can be classified into two main categories namely *avoidance methods* and *tolerance methods* [3]. The first try to exploit certain properties of the net to reduce the number of states of the underlying CTMC. Of course they can only be applied as far as the demanded properties are inherent in the modeled system. Methods of the second category try to tolerate the large state spaces. One way to do this is the parallelization of the analysis [4,6,14,2]. Tolerance methods may be seen as additional possibilities to be applied side by side with avoidance methods or other tolerance methods.

The most difficult part of the numerical analysis process to parallelize is the unfolding of the state space that defines the sparse matrix of a CTMC [5,6]. This is a graph generation process starting with just an initial state. The difficulties for a parallelization lie in the irregular, dynamically growing data structure of the CTMC whose size and shape cannot be estimated in advance.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 147–167, 1999.
© Springer-Verlag Berlin Heidelberg 1999

After the CTMC is generated, the transient numerical solver computes its state probabilities for the desired time points. Our algorithms use the uniformization method [19,17,18]. Uniformization is a very popular transient solution method as it shows good convergence behavior in many cases, is memory efficient and easy to implement.

The last step in the quantitative analysis process is the back-transformation of the results to the net level. For this purpose stochastic reward nets (SRNs) [5] provide methods to impose *rate rewards* on the states (gained whenever time is spent in these) and *impulse rewards* on the transitions of the CTMC (gained if this state transition occurs).

Our Contribution. We are looking at the theoretically well-defined transient analysis process from the realization point of view, discuss design decisions and show the difficulties that arise within the numerical analysis process especially for the parallelization approaches.

Algorithms for distributed-memory (DM) machines implemented with the common available message-passing libraries PVM (Parallel Virtual Machine) [11] or MPI (Message Passing Interface) [10] are a cheap way to utilize the main memories of clustered workstations or PCs which are maintained in nearly every office nowadays. In our design decisions we focused on the space problem rather than the computation time. This means in contrast to existing approaches to the state space generation [6,14] we implemented dynamical load balancing algorithms which cause computational overhead on one hand but guarantee on the other hand that the computation is not to be aborted prematurely due to a lack of memory on a subset of the machines while other CPU's local memories remain widely unused. Furthermore with this approach we are free to automatically enhance the number of processors during runtime if necessary.

In contrast to existing parallel approaches our shared-memory (SM) algorithms perform no partitioning of the state space onto the processors but implement suitable synchronization schemes on the global shared data. It can be shown that good speedups are reached during the whole analysis process on these architectures.

As far as we know there exist no parallel transient analysis solvers for GSPNs or SRNs in the literature up to now. In the shared-memory case the parallel realization of the uniformization method and the result measure computation is straight forward. In the distributed-memory version the scattered state space imposes problems. Nevertheless we can show that computation times do not degrade relatively to the single processor version and for models that touch the limits of a single processor machine enormous acceleration is achieved by the elimination of swapping effects. This way sequentially not manageable problem sizes can be solved combining the main memories of clustered workstations. Computation time and manageable problem size is compared with the sequential GSPN tool SPNP (Stochastic Petri Net Package) [8].

As an elaborated result measure concept was designed, all the algorithms described could be integrated to build up the transient numerical analysis component of the SRN modeling tool ℙANDA . ℙANDA [1] provides a graphical

user interface allowing the specification of GSPNs expanded with phase-type distributed firing times. A widely SPNP [8] compatible C-language interface[1] can also be used. This means the parallel concepts can be utilized in a flexible, portable and comprehensive way.

2 State Space Generation

The first step in the numerical analysis process of GSPNs is the state space generation [16,7]. During this phase the CTMC is constructed in an iterative way starting from the initial marking $\boldsymbol{m}_0$ of the GSPN. The states $\boldsymbol{m}_i \in \mathcal{M}$ of the CTMC are the tangible markings of the GSPN (markings which enable timed GSPN transitions), an arc $a_k^{i \to j} \in \mathcal{A}$ of the CTMC graph can be associated with the firing of a certain timed transition t of the GSPN leading from marking $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ denoted by $\boldsymbol{m}_i \xrightarrow{t} \boldsymbol{m}_j$. To be able to assign impulse rewards (Sec. 4) to the timed GSPN transitions each arc $a_k^{i \to j}$ is labeled with the transition ID $l_k = x$ of the transition $t_x \in \mathcal{T}$ whose firing led to $a_k^{i \to j}$ and its possibly marking dependent firing rate $r_k = r(t_x, \boldsymbol{m}_i)$. Multiple arcs between two states of the CTMC labeled with different transition IDs may not be combined into one in order to preserve the correlation between GSPN transitions and CTMC arcs.

In our algorithms vanishing markings — i.e. markings which enable immediate transitions — are eliminated on-the-fly [6] as it is the method of choice for a transient analysis [5] although this impedes the computation of impulse rewards for immediate transitions.

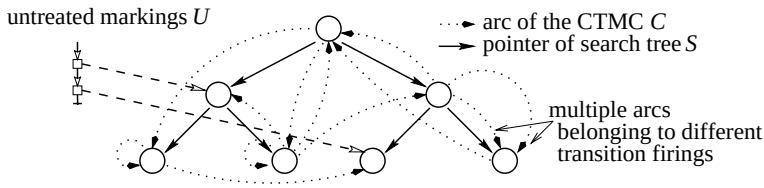


Fig. 1. Data structures of the CTMC generation.

Data Structures. Mainly three dynamically changing data structures determine the algorithms: the accruing CTMC graph C, the search data structure S to retrieve already created states for not being inserted several times into C and a set of untreated markings U to temporarily store markings until their successor markings in C are generated. U and S are needed during CTMC generation only. We realized U as linear list of pointers, pointing to states already inserted in the CTMC C (for alternatives see [7]).

[1] The definition of the result measures differs from CSPL (C-based Stochastic Petri Net Language) as impulse rewards are allowed.

S is the crucial data structure with respect to efficiency because it is consulted at least once for each arc of the CTMC. We prefer balanced search trees [13] over hash tables [13] to implement S because their behavior in the state space generation context is more predictable in both memory consumption and search overhead. A hash table is either based on heuristics to determine its size or must use complex dynamic expansion techniques. It may not be simply as big as possible to retrieve and store the states of the CTMC as the arcs will also occupy an unpredictable amount of memory. The hash function is also heuristic and must therefore handle collisions appropriately. Since the numerical uniformization solution needs five additional floating point values per CTMC state anyway (see Sec. 3) implementing S as a search tree does not cause a bottleneck with respect to memory consumption as the memory of the child pointers of S is used to store these floating point values after the CTMC generation is finished.

In algorithmic considerations we often treat the entries of S and the states of C like different elements. But an implementation will only maintain data structures for the states of the CTMC and augment them with child pointers. Fig. 1 illustrates the connections between C, S and U.

Algorithmic Aspects. Fig. 2 and 3 show a sketch of the CTMC generation algorithm in pseudo-code. The lines marked with $*$ are parallelization specific, omitting them results in the sequential algorithm we will regard first. Procedure `CTMC_generation()` Fig. 2 is the main routine. Realizing U as linear list implies that `take_element()` (Line 15) just takes the first element of the list which equals a FIFO stategy leading to a breadth-first traversal of the CTMC graph.

New states and arcs are inserted into the accruing CTMC by procedure `new_state()` Fig. 3. Arcs $a_k^{i \to j}$ are stored with their destination state $\boldsymbol{m}_j$ which is essential for both the SM and the DM parallelization of the CTMC generation. The elimination on-the-fly is done by procedure `elim_otf()` Fig. 3. The degree of parallelization is increased by the on-the-fly elimination because the global data structures S, C and U do not need to be consulted for vanishing states and therefore they cause no communication or synchronization.

For simplicity it is assumed that the initial marking is a tangible one. This restriction can be released by adding a simple initialization routine [5]. The on-the-fly elimination is specified as a recursive procedure (Line 51). As recursion depth is a limiting factor the actual realization is iterative using a stack.

The search key which is looked for is the marking itself which is simply an integer vector whose components are the number of tokens in the corresponding GSPN places. The lexicographical order is imposed on the markings so that the tree can be traversed comparing the marking met to the one that is looked for.

Storage Requirements. Each state stores: (1) *A marking* that is either a vector of fixed size which is advantageous for Petri nets where all the places contain tokens most of the time, or of variable size storing both the place number and the number of tokens for those places that contain at least one token. Second is the method of choice for most of the nets. (2) *An identification number.* This is only for efficiency reasons and could be omitted if the marking was used to identify

```
* 1 private process ID p
  2 initial marking m_0
  3 CTMC C = ∅
  4 search structure S := ∅
  5 set of untreated markings U := ∅

  6 procedure CTMC_generation( )
  7 begin
* 8    if (partition(m_0) = p)
  9       insert m_0 in C
 10       insert m_0 in S
 11       U := U ∪ m_0
*12    endif
*13    while (no termination message received)
 14       while (U ≠ ∅)
 15          m_f := take_element(U)
 16          for each enabled transition t_x : m_f --t_x--> m_c
 17             if (m_c is vanishing)
 18                elim_otf(m_c, x, r(t_x, m_f), f)
 19             else
 20                new_state(m_c, x, r(t_x, m_f), f)
 21             endif
 22          endfor
*23          offer()
 24       endwhile
*25       request()
*26    endwhile
 27 end CTMC_generation
```

Fig. 2. Main routine of the parallel CTMC generation algorithm for DM machines. Omitting the $*$-marked lines results in the sequential algorithm.

a state. (3) *A pointer to the incoming arcs* within C. (4) *Pointers to construct* S. The structure of S can be converted after CTMC generation to connect all the states in a linear list and unnecessary pointers are freed or reused otherwise. Each arc stores: (1) *A transition ID.* (2) *A rate value.* (3) *A link to its source state.* If arcs are not maintained in a dynamically growing array also (4) *a pointer to the next arc* must be given otherwise the number of incoming arcs has to be stored. Up to now our implementations do unfortunately not implement these structures at minimum cost. U represents no limiting factor. If the elements $u_i \in U$ point to states which are not absorbing, at least one arc — consuming more memory than u_i — will be inserted in C when u_i is deleted (Line 15 Fig. 2 and Line 38 Fig. 3). Therefore the memory consumed by U is finally needed anyway for C if we assume only few absorbing states.

```
 28 procedure new_state(m_c, x, rate, f)
 29 begin
*30    n := partition(m_c)
*31    if (n = p)
 32       if (m_c ∉ S)
 33          insert m_c in C
 34          insert m_c in S
 35          U := U ∪ m_c
 36       endif
 37       if (not exists arc a_k^{f→c} in C with l_k = x)
 38          insert a_k^{f→c} in C
 39          set transition label l_k := x and rate label r_k := 0
 40       endif
 41       set rate label r_k := r_k + rate
*42    else
*43       pack(n, m_c, x, rate, f)
*44    endif
 45 end new_state

 46 procedure elim_otf(m_v, x, rate, f)
 47 begin
 48    for each enabled transition t_y : m_v --t_y--> m_c
 49       rate := rate * r(t_y, m_v)
 50       if (m_c is vanishing)
 51          elim_otf(m_c, x, rate, f)
 52       else
 53          new_state(m_c, x, rate, f)
 54       endif
 55    endfor
 56 end elim_otf
```

Fig. 3. Subroutines of the parallel CTMC generation algorithm for DM machines. Omitting the $*$-marked lines results in the sequential procedures.

2.1 Distributed-Memory Parallelization

The main difficulty of the DM approach is to achieve an equal distribution of the state space onto processors, since each processor has access only to its local memory and the part of the state space stored there. Since there is usually no information available about the state space before it is actually built, the partitioning function should be adapted during run time. A dynamically changing partitioning function is the main difference to existing approaches in [6,14].

Algorithmic Aspects. We use a master/slave concept: one slave process runs on each processor. The slaves are concerned with the actual state space generation given in Fig. 2 and 3. Additionally one master process exists. It is responsible only for the spawning of the slaves and further on for administering, controlling and

terminating them. We do not give an algorithmic sketch of the master process as it is just reacting on messages from the slaves which it attends in a loop.

Four routines are added to the sequential algorithm:

- `partition(`$\boldsymbol{m}_c$`)` returns the process ID n of the slave which is responsible for marking $\boldsymbol{m}_c$. This way each state is mapped uniquely onto one processor and this processor finds the marking in its search tree if it is already generated.
- `pack(`n`,` $\boldsymbol{m}_c$`,` x`,` $rate$`,` f`)` packs marking $\boldsymbol{m}_c$ together with the information needed to create a new arc into the output buffer for messages that wait to be sent to the slave with ID n.
- `request()` is called if a slave gets idle, i.e. when its U got empty. All of the output buffers are sent to the corresponding slaves and a idle message is broadcasted. If new markings arrive they are handeled by calling `new_state()` (giving the received additional information as arguments). Furthermore attention is given to load balancing and termination messages.
- `offer()` is a cyclically called routine that checks for data requests of other slaves. If one is queued the corresponding output buffer is sent. In order to keep input buffers small procedure `new_state()` is called for each so-far received marking. Again attention is given to load balancing and termination messages. Procedure `offer()` (Line 23, Fig. 2) should only be called every 100th iteration.

Dynamic Load Balancing. Our solution to the partitioning problem is a dynamic load balancing algorithm. Hereby memory consumption and not computation time is regarded as load. The load balancing routine repartitions the CTMC and adapts the partitioning function.

The partitioning function of course maps the entire set of possible markings onto the processors. Potential markings are ordered lexicographically again and each slave is assigned a continuous (not necessarily equal sized) range of these. The markings which build up the borders of the ranges are known to all the slave processes. `partition(`$\boldsymbol{m}_c$`)` (Line 30, Fig. 3) has to compare the current marking $\boldsymbol{m}_c$ with the range borders and return the process ID n of the slave whose border markings $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ include $\boldsymbol{m}_c$: $\boldsymbol{m}_i \leq \boldsymbol{m}_c \leq \boldsymbol{m}_j$. To do this efficiently the border markings are organized in a binary tree instead of a linear list. The ranges are recomputed when load is rebalanced.

The master process is cyclically informed by the slaves how much of their local memory is occupied. When the memory utilization on one processor differs more than $m\%$ from the average memory utilization (tolerance parameter m can be chosen), load balancing is initiated by the master. For the load balancing routine a chain topology is used. The master tells each slave how many bytes it has to send to or get from its left and right neighbor processor. Then the slaves exchange CTMC states together with their incoming arcs in parallel until the required amount of bytes is exchanged. This way also the arcs of the CTMC are considered when memory is balanced which we experienced to be crucial (see Sec. 5). To maintain the continuous ranges of the partitioning function the right neighbor gets the smallest markings and the left neighbor gets the biggest ones.

Fig. 4 illustrates the load balancing. For simplicity markings are represented

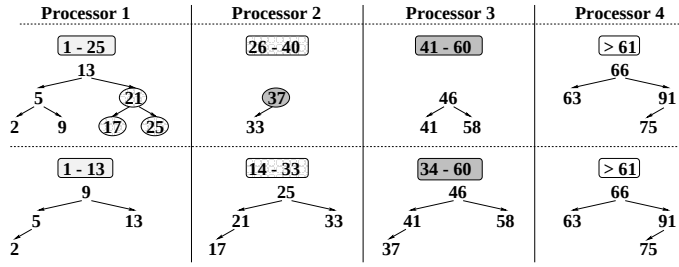


Fig. 4. Load balancing mechanism.

as integer numbers instead of vectors. It shows the ranges of four processors with their local balanced search trees S. The upper line shows the situation before load balancing, the lower line afterwards. Processor 1 has to send three markings to Processor 2 (as indicated by the fill style) and Processor 2 has to send one to Processor 3. The trees are rebalanced during the exchange.

We implemented S by an AVL-tree [13] in our DM algorithm which is a height-balanced binary tree. Rebalancing is done by rotating smaller or bigger parts of the tree thus guaranteeing search times that are logarithmic in the number of the so-far generated states. The balance of the AVL-tree is maintained easily during load balancing as always the smallest/biggest marking is removed from the tree. This is simply done by cutting off the leftmost/rightmost leaf and initiating a rotation to rebalance the tree immediately if necessary. The destination process receives the markings in an ordered way which makes the insertion on receiver's side as simple as the cutoff on sender's side. When the exchange is finished the slaves report the number of markings exchanged and the master process recomputes the new range borders and broadcasts them.

In contrast to the shared-memory version the arc $a_k^{i \to j}$ that is stored with its destination state $\boldsymbol{m}_j$ cannot contain a pointer to its source state $\boldsymbol{m}_i$ because $\boldsymbol{m}_i$ may be stored on a different processor. Instead of pointers it just contains the unique source state ID which consists of process ID and state number. After state space generation is finished a renumbering has to be performed to prepare the numerical solution: an unique integer number is computed out of the bipartite number and the numbers are permuted (as they are disordered due to load balancing) so that each processor p ends up with a continuous range of state numbers $\varrho_p = [\varrho_p^{\triangleright}, \varrho_p^{\triangleleft}]$ in its memory. This means the source state entries of the arcs have to be changed also. Computation times for renumbering are negligible.

Additional Storage Requirements. DM parallelization additionally needs space for input and output buffers to store the data sent to/received from other processors. The buffers may contain one marking several times. The state ID has to store also the process number additionally to the state number. AVL-trees are only efficient if a parent pointer is maintained for each state.

2.2 Shared-Memory Parallelization

When a shared-memory architecture is used, the main data structures C, S and U (Fig. 1) may be kept in global shared memory. This means no partitioning onto the processors has to be performed and the current information about the so-far generated state space is accessible for all the processors.

To guarantee data consistency it is necessary to lock data portions using mutex variables so that read and write operations are done in mutual exclusive access. Otherwise two processors might for example insert the same state in the CTMC at the same time. Mutex variables are tested and set in an atomic operation. Only a thread[2] which succeeded in setting the mutex variable may proceed whereas the other threads block until the mutex variable is unlocked again. To allow a high degree of parallelism the portion of the global data that is locked by one processor should be small and should not represent a bottleneck. But on the other hand the locked data portions should not be too small as locking operations are expensive and locking variables consume memory.

The main problem with balanced search trees and synchronization is the rebalancing routine: insertions at a leaf of the tree often cause reorganizations which may affect the whole tree structure. Therefore the part of the tree that is potentially affected by the reorganization has to be locked.

In our solution data consistency with a high degree of parallelism is achieved by using B-trees [9] as search data structure S. In B-trees a parameterizable number of markings is grouped together to one B-tree node which is the entity that is locked thus reducing the number of locking operations. Rebalancing is done by splitting B-tree nodes when they are full. The height of the tree is only increased when the root node is split. Normally rebalancing is done in a back-propagation manner: the insertion of a new marking in a leaf node causes it to split when full giving one marking to its parent node. This again may be full and has to be split and so on. This way splittings may propagate back up to the root node which is very disadvantageous for synchronization. We therefore modified the standard balancing strategy from splitting nodes on demand to a splitting-in-advance strategy: nodes which perhaps get full when the next insertion takes place are split in advance on the way down during the search process. This strategy works with only two mutex variables locked at a time per thread. As there is no room for details we refer to [2] for a full description.

Additional Storage Requirements. The SM algorithm needs one mutex variable per B-tree node which is $O(|\mathcal{M}|/b)$ on an average (with b average node size), but no parent pointers have to be maintained. State numbers are augmented by the thread number.

3 Transient Numerical Solution

Now as the CTMC is generated, *instantaneous* [19] and *cumulative* [18] transient measures are computed for one or a series of user-defined time points.

[2] processes are called threads in the SM context

For the basic theoretical considerations we switch from regarding the CTMC as a graph (which is nothing else than a sparse matrix representation) to the full matrix representation. This means vector component $\pi_i(\tau)$ of the vector $\boldsymbol{\pi}(\tau)$ represents the state probability of CTMC state $\boldsymbol{m}_i$ at time τ and component $\theta_i(\tau)$ of vector $\boldsymbol{\theta}(\tau)$ represents the expected total time that is spent in state $\boldsymbol{m}_i$ during time interval $[0, \tau]$. The infinitesimal generator matrix $\boldsymbol{Q}$ is built up of the transition rates labeling the arcs of the CTMC graph:

$$q_{i,j} = \sum_{a_k^{i\to j}\in\mathcal{A}} r_k, \qquad \text{with } i \neq j\ , \qquad q_{i,i} = -\sum_{a_k^{i\to j}\in\mathcal{A}, j\neq i} r_k\ . \tag{1}$$

Uniformization Algorithm. The Chapman-Kolmogorov equation of the time homogeneous CTMC — given the initial state probability vector $\boldsymbol{\pi}(0)$ — is $\frac{\delta\boldsymbol{\pi}(\tau)}{\delta\tau} = \boldsymbol{\pi}(\tau)\boldsymbol{Q}$. We have to set $\pi_0(0) = 1$ for the initial marking $\boldsymbol{m}_0$ and $\pi_i(0) = 0$ for $i \neq 0$.[3] The uniformization method [19,5,17] computes the solution of the Chapman-Kolmogorov equation by evaluating

$$\begin{aligned}\boldsymbol{\pi}(\tau) \approx \boldsymbol{\pi}^*(\tau) &= \sum_{h=\triangleright(q\tau,\epsilon)}^{\triangleleft(q\tau,\epsilon)} \underbrace{\boldsymbol{\pi}(0)\,\hat{\boldsymbol{Q}}^h}_{\boldsymbol{\varphi}(h)} \underbrace{e^{-q\tau}\frac{(q\tau)^h}{h!}}_{\beta(h,q\tau).}\ , \\ &= \sum_{h=\triangleright(q\tau,\epsilon)}^{\triangleleft(q\tau,\epsilon)} \boldsymbol{\varphi}(h)\qquad \beta(h,q\tau).\end{aligned} \tag{2}$$

where factor q is given by $q \geq \max_i |q_{i,i}|$ and Matrix $\hat{\boldsymbol{Q}}$ is defined by $\hat{\boldsymbol{Q}} = \frac{1}{q}\boldsymbol{Q} + \boldsymbol{I}$ with $\boldsymbol{I}$ being the identity matrix. To satisfy $||\boldsymbol{\pi}(\tau) - \boldsymbol{\pi}^*(\tau)||_\infty \leq \epsilon$ we have to set (see [19,5]) $\triangleright(q\tau,\epsilon) = \max\{j \in \mathbb{N} : \sum_{h=0}^{j-1} \beta(h, q\tau) \leq \frac{\epsilon}{2}\}$ and $\triangleleft(q\tau,\epsilon) = \min\{j \in \mathbb{N} : 1 - \sum_{h=0}^{j} \beta(h, q\tau) \leq \frac{\epsilon}{2}\}$.

The real work lies in the vector/matrix multiplications computing $\boldsymbol{\varphi}(h)$ (Eq. 2) which is done iteratively:

$$\begin{aligned}&\boldsymbol{\varphi}(0) = \boldsymbol{\pi}(0) \\ &\boldsymbol{\varphi}(h) = \boldsymbol{\varphi}(h-1)\,\hat{\boldsymbol{Q}}, \qquad\qquad h > 0\ .\end{aligned} \tag{3}$$

Restarted Version. We realized the restarted version of the algorithm given in [19]. This means if several time points $\tau_0, ..., \tau_{\sharp\tau}$ have to be evaluated, for τ_i with $0 < i \leq \sharp\tau$ we set $\tau = \tau_i - \tau_{i-1}$ and $\boldsymbol{\pi}(0) = \boldsymbol{\pi}^*(\tau_{i-1})$ in Eq. 2 and start again. This means we assign each time interval $i > 0$ its own formula ${}^i\boldsymbol{\pi}^*(\tau)$ with

$$\boldsymbol{\pi}^*(\tau_i) = {}^i\boldsymbol{\pi}^*(\tau_i - \tau_{i-1}) = \sum_{h=\triangleright(q(\tau_i-\tau_{i-1}),\epsilon)}^{\triangleleft(q(\tau_i-\tau_{i-1}),\epsilon)} \boldsymbol{\pi}^*(\tau_{i-1})\,\hat{\boldsymbol{Q}}^h\ \beta(h, q(\tau_i - \tau_{i-1}))\ . \tag{4}$$

This is possible because of the Markov property: the past has no influence on the future so it can be assumed that time point τ_{i-1} is the initial time. As

[3] If the initial marking is vanishing several $\pi_i(0) > 0$ may exist. This initial vector $\boldsymbol{\pi}(0)$ must be preserved by the CTMC generation.

errors sum up this way the internal error tolerance must be set to $\epsilon' = \frac{\epsilon}{\sharp\tau}$ for the computation of each time point. Restarting causes overhead in computation time because for each time point

- the left truncation point $\triangleright(q\tau, \epsilon)$ has to be reached;
- since holding the two matrices $\boldsymbol{Q}$ and $\hat{\boldsymbol{Q}}$ in memory is not acceptable with respect to storage requirements, matrix $\boldsymbol{Q}$ has to be converted to $\hat{\boldsymbol{Q}}$ and back again to compute result measures;
- for each user-defined result measure function every state and arc has to be visited in the worst case as we will see in the next section. Without restart all $\sharp\tau$ time points were computed in one step and stored in an array of size $\sharp\tau$ within the states. Therefore the result computation would have to visit the states and arcs only once, calling each result function only once (implying that the result functions operate on an array of size $\sharp\tau$).

But as we consider memory consumption as the limiting factor, we accept this computational overhead (which we experienced to be negligible) and just store one time point at a time, doing result computation straight ahead and then restart for the next time point.

Expected Total Time. The vector of the expected total times spent in the states $\boldsymbol{\theta}(\tau) = \int_0^\tau \boldsymbol{\pi}(u)\, du$ is computed by [18]

$$\boldsymbol{\theta}(\tau) \approx \boldsymbol{\theta}^*(\tau) = \int_0^\tau \boldsymbol{\pi}^*(u)\, du = \frac{1}{q} \sum_{h=0}^{\lhd(q\tau,\epsilon)} \boldsymbol{\pi}(0)\hat{\boldsymbol{Q}}^h \left(1 - \sum_{i=0}^{h} e^{-q\tau} \frac{(q\tau)^i}{i!}\right) . \quad (5)$$

Time averaged values $\overline{\boldsymbol{\theta}}(\tau) = \boldsymbol{\theta}(\tau)/\tau$ are within the error tolerance ϵ this way, for absolute values we set $\epsilon = \epsilon/\tau$ and adapt $\lhd(q\tau, \epsilon)$. As ${}^i\boldsymbol{\pi}^*(\tau) = {}^{i-1}\boldsymbol{\pi}^*(\tau + \tau_{i-1})$ for $\tau > \tau_{i-1}$ which means that restart $i > 0$ shifts function ${}^{i-1}\boldsymbol{\pi}^*(\tau)$ τ_{i-1} units to the left, we may also compute $\boldsymbol{\theta}^*(\tau)$ together with $\boldsymbol{\pi}^*(\tau)$ in a restarted manner:

$$\boldsymbol{\theta}^*(\tau_i) = {}^i\boldsymbol{\theta}^*(\tau_i - \tau_{i-1}) = {}^{i-1}\boldsymbol{\theta}^*(\tau_{i-1}) + \int_0^{\tau_i - \tau_{i-1}} {}^i\boldsymbol{\pi}^*(u)\, du\,. \quad (6)$$

Storage Requirements. For each state $\boldsymbol{m}_i \in \mathcal{M}$ of the CTMC $\boldsymbol{\varphi}(h)$ and $\boldsymbol{\varphi}(h-1)$, the instantaneous and cumulative solution and the diagonal entry of $\hat{\boldsymbol{Q}}$ have to be stored. This results in an additional storage expenditure of $|\mathcal{M}| * 5$ floating point values for the transient uniformization solution.

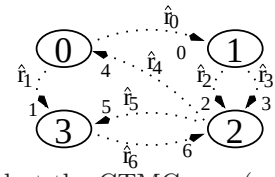
3.1 Distributed-Memory Parallelization

The conversion of matrix $\boldsymbol{Q}$ to matrix $\hat{\boldsymbol{Q}}$ is trivial to parallelize. The diagonal elements $q_{i,i}$ are already computed during CTMC generation. The local maximum $q_{i,i}$ element is determined and given to the master which then determines the

global maximum and gives it to the slaves which in turn compute their part of $\hat{\boldsymbol{Q}}$. The truncation points and $\beta(h, q\tau)$ are computed locally on each processor.

The parallelization issue is consequently the vector/matrix multiplication to compute $\boldsymbol{\varphi}(h)$ in each iteration h (Eq. 3) where we have to cope with the scattered state space. The matrix $\hat{\boldsymbol{Q}}$ is stored columnwise by storing the incoming arcs with the CTMC states as described in Sec. 2. The renumbering of the states (Sec. 2.1) additionally implies that the locally stored state numbers build a continuous range which is the only way to allow $\boldsymbol{\varphi}(h)$ and $\boldsymbol{\varphi}(h-1)$ being stored in arrays and accessing their components via the array index evaluated from the state number. This in turn is essential as we will see, since $\phi_i(h-1)$ has to be referenced via the information stored in $a_k^{i\to j}$ which provides only the source state ID i. Fig. 5 illustrates these data structures for matrix

$$\hat{\boldsymbol{Q}} = \begin{pmatrix} \hat{q}_{0,0} & \hat{r}_0 & 0 & \hat{r}_1 \\ 0 & \hat{q}_{1,1} & \hat{r}_2 + \hat{r}_3 & 0 \\ \hat{r}_4 & 0 & \hat{q}_{2,2} & \hat{r}_5 \\ 0 & 0 & \hat{r}_6 & \hat{q}_{3,3} \end{pmatrix} \quad \Leftarrow$$



where $\hat{r}_i = r_i/q$ are the converted rate values stored at the CTMC arcs (see Eq. 1). The data structure of an incoming arc containing its rate value and source state number is given as the rectangle in Fig. 5. In order to be really scalable we implemented two possibilities: (1) a more time efficient one-step vector/matrix multiplication which requires storing a vector of size $|\mathcal{M}|$ on each processor and (2) a multiplication in $\sharp p$ steps ($\sharp p$ number of processors) using a smaller array of size x, x being the maximum number of states on one processor. We describe the second method first: Two steps are illustrated by Fig. 5. In the first step of

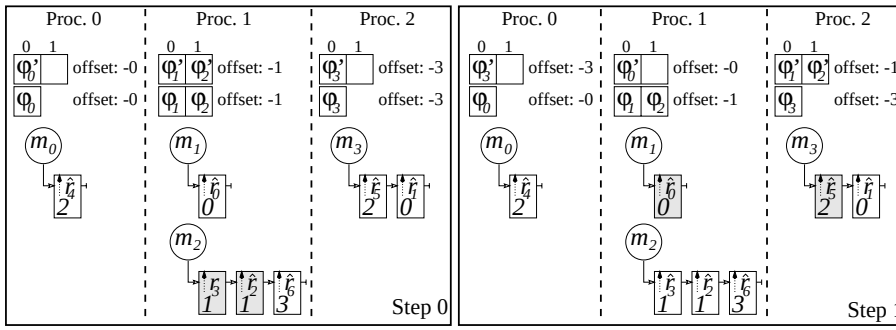


Fig. 5. Two steps of parallel distributed-memory vector/matrix multiplication.

iteration h each processor p updates the $\boldsymbol{\varphi}' = \boldsymbol{\varphi}(h-1)$ array in Fig. 5 which means $\boldsymbol{\varphi}' := \boldsymbol{\varphi}$ and visits all of its local arcs $a_k^{i\to j}$ with $i \neq j$ summing up

$$\varphi_j(h) = \varphi_j = \sum_{a_k^{i\to j} \in \mathcal{A}, i \in \varrho_p, i \neq j} \hat{r}_k * \varphi'_i \,. \tag{7}$$

This can be done only for arcs with $i \in \varrho_p$, ϱ_p being the range of state numbers of processor p for which the $\boldsymbol{\varphi}'$ values are currently available in the $\boldsymbol{\varphi}'$-array. In Fig. 5 the filled style marks for which arcs $a_k^{i \to j}$ the products $r_k * \varphi'_i$ can be summed up to φ_j. The left border $\underline{\varrho}_p^{\triangleright}$ of ϱ_p builds the offset for both arrays: The index of $\boldsymbol{\varphi}'$ is given by $i - \underline{\varrho}_p^{\triangleright}$ that of $\boldsymbol{\varphi}$ by $j - \underline{\varrho}_p^{\triangleright}$. After each step $s = 0, ..., \sharp p - 1$ the $\boldsymbol{\varphi}'$-arrays are moved in a circle by giving the array to the left neighbor processor with processor number $(p + 1) \bmod \sharp p$. This way each processor $n = (p + s) \bmod \sharp p$ computes Eq. 7 in Step s for $i \in \varrho_p$ by reaching φ'_i in array component $i - \underline{\varrho}_p^{\triangleright}$. After $\sharp p$ steps $\boldsymbol{\varphi} = \boldsymbol{\varphi}(h)$ is totally computed.

The one-step multiplication on the other hand just has two phases: first each processor sends its local part of $\boldsymbol{\varphi}'$ to all the other processors. Every processor stores the whole $\boldsymbol{\varphi}'$ vector. Therefore in the second phase the processors can locally update their $\boldsymbol{\varphi}$ values in one step.

3.2 Shared-Memory Parallelization

Parallelizing the transient uniformization solver on a shared-memory machine is quite simple. During the whole computation each thread p is assigned a subset of states $\mathcal{M}_p$ with $\bigcup_p \mathcal{M}_p = \mathcal{M}$, $\mathcal{M}_p \cap \mathcal{M}_q = \emptyset$ and $|\mathcal{M}_p| \approx |\mathcal{M}_q|$ for $p \neq q$. All the CTMC states reside in global shared memory and are therefore all accessible for all the processors. Synchronization can be limited to the insertion of some barriers, at which the threads stop until all of them have reached this point.

Barriers are needed to compute the stochastic matrix $\hat{\boldsymbol{Q}}$ out of matrix $\boldsymbol{Q}$: first the diagonal elements $q_{i,i}$ are computed in parallel. During CTMC generation this is not possible since only the destination state $\boldsymbol{m}_j$ of a newly generated arc $a_k^{i \to j}$ is protected by a mutex variable and the $q_{i,i}$ computation would change source state $\boldsymbol{m}_i$ for $a_k^{i \to j}$. The computation is split into $\sharp p$ phases ($\sharp p$ number of processors). In each phase $s = 0, ..., \sharp p$ each thread $p \in [0, \sharp p]$ computes thread number $n = (p + s) \bmod \sharp p$. Then *all* the incoming arcs of each of the thread's states $\boldsymbol{m}_j \in \mathcal{M}_p$ are visited but the value r_k of the current arc $a_k^{i \to j}, i \neq j$ is only added to $q_{i,i}$ if $\boldsymbol{m}_i \in \mathcal{M}_n$. This way each thread works on a different set of source states $\mathcal{M}_n$ in each phase and no data protection has to be done. The phases have to be separated by barriers. Afterwards the maximum $q_{i,i}$ can be found for each thread's subset of states in parallel followed by a barrier to wait until the global maximum is determined. Dividing all matrix entries to gain $\hat{\boldsymbol{Q}}$ is done completely in parallel without synchronization.

For the vector/matrix multiplication (Eq. 3) no synchronization is needed because of the columnwise storage scheme: $\varphi_j(h) = \sum_{a_k^{i \to j} \in \mathcal{A}} \hat{r}_k * \varphi_i(h-1),\ i \neq j$ is computed by each thread p for all the states $\boldsymbol{m}_j \in \mathcal{M}_p$ referencing $\varphi_i(h-1)$ (which is stored within state $\boldsymbol{m}_i$) by the source state pointer within $a_k^{i \to j}$. After each iteration a barrier has to be inserted.

4 Reward Based Result Measure Computation

In this section we focus on implementation issues of a flexible result measure definition and evaluation. Result measures are given by the modeler as a part of the net definition which lifts the GSPN up to a SRN [5]. For ℙANDA in which we integrated all the parallel algorithms this can be done either within the graphical interface (which imposes some restrictions on the functions as an interpreter has to provide the operators) or using the SPNP similar C-interface [8] where the flexibility of the C-language may be exploited. Result measures are specified by

1. first defining rate and impulse reward functions [5]
2. and then combining a pair of one rate and one impulse reward thus defining — together with the CTMC — a global reward process $\mathcal{R}(\tau)$ and computing a stochastic value like the expected global reward $E[\mathcal{R}(\tau)]$. The wanted result measure may simply be this stochastic value itself or arithmetic and logical operators can be applied to combine several stochastic values to one result measure [12]. Several result measures may be specified within one net.

In Step 1 the user needs a means to assign the rate reward ρ_i to state $\boldsymbol{m}_i$ and the impulse reward ι_k to arc a_k of the CTMC by defining rate and impulse reward functions ρ and ι on SRN level. The *characterizing functions* CF are the building blocks of the user-defined reward functions and are to be provided by the tool. We distinguish the *node characterizing functions* NCF

- `mark`(p_x): $\mathcal{P} \times \mathcal{M} \to \mathbb{N}_0$ returning the number of tokens in place $p_x \in \mathcal{P}$ in the current marking $\boldsymbol{m}_i \in \mathcal{M}$,
- `enabled`(t_y): $\mathcal{T} \times \mathcal{M} \to \{0, 1\}$ returning 1 if transition $t_y \in \mathcal{T}$ is enabled in the current state $\boldsymbol{m}_i \in \mathcal{M}$ and 0 if not

and the *arc characterizing functions* ACF

- `fire`(t_y): $\mathcal{T} \times \mathcal{A} \to \{0, 1\}$ returning 1 if the firing of transition $t_y \in \mathcal{T}$ is responsible for the current arc $a_k \in \mathcal{A}$ (i.e. $l_k = y$) and 0 if not,
- `rate`(t_y): $\mathcal{T} \times \mathcal{A} \to \mathbb{R}$ returning the firing rate r_k of transition $t_y \in \mathcal{T}$ for the arcs $a_k^{i \to j} \in \mathcal{A}$ with transition label $l_k = y$, 0 for the others.

The user fixes the place and transition parameters of the CF which means the resulting reward function is a function of the current CTMC state or arc only. For rate rewards we allow only NCF to be used. For impulse rewards ACF and NCF are available in the sequential and shared-memory context. In the distributed-memory context only ACF are allowed for impulse rewards since otherwise we would get into troubles as we will see in Sec. 4.1.

Within the result computation self-arcs within the state space should not be neglected because an impulse reward may be assigned to them. In the numerical solution they are not included in matrix $\boldsymbol{Q}$ (see Eq. 1) as transition firings which do not change the marking are not regarded as iterruption of the sojourn in this marking. But nevertheless the CTMC generation preserves them (see algorithm

Figures 2 and 3) in the arc lists so they can be treated just like any other arc during result computation (compare [5]).

In this first parallel PANDA version the only stochastic operator that is provided is the expected value E. The expected value of the instantaneous reward at time τ is given by

$$E[\mathcal{R}'(\tau)] = \sum_{\boldsymbol{m}_i \in \mathcal{M}} \rho_i \; \pi_i(\tau) + \sum_{a_k^{i \to j} \in \mathcal{A}} \iota_k \; r_k \; \pi_i(\tau) \tag{8}$$

and the expected value of the reward up to time τ

$$E[\mathcal{R}(\tau)] = \sum_{\boldsymbol{m}_i \in \mathcal{M}} \rho_i \; \theta_i(\tau) + \sum_{a_k^{i \to j} \in \mathcal{A}} \iota_k \; r_k \; \theta_i(\tau) \tag{9}$$

(see [5] for details). This means the algorithm has to visit each state $\boldsymbol{m}_i$ of the CTMC and call the user-defined rate reward function to get ρ_i and each arc a_k and call the impulse reward function to get ι_k. Regarding each arc separately for impulse rewards is necessary since functions like $\texttt{ifn}() = 2\,\texttt{fire}(t_0) + 3\,\texttt{fire}(t_1)$ may be user-defined and arcs $a_k^{i \to j}$ and $a_h^{i \to g}$ with transition labels $l_k = 0$ and $l_h = 1$ may both origin from source state $\boldsymbol{m}_i$. Therefore just calling `ifn()` once for $\boldsymbol{m}_i$ is not possible.

Rate and impulse rewards could have been most efficiently assigned to the states/arcs of the CTMC during CTMC generation itself by just calling each user-defined rate/impulse reward function once for each newly generated state/updated arc. But then (1) possibly several reward values would have to be stored with the states/arcs of the CTMC consuming memory until the result computation could evaluate the stochastic values. Alternatively (2) the reward values can be computed during CTMC generation and immediately be written to an external file to stay there until they are needed. But writing and reading files is time consuming. Therefore we decided (3) to compute rewards in a third step together with the result measure evaluation. As the measurements show, the complete result measure computation is relatively fast compared to the CTMC generation and numerical solution. If the limitation to ACF for impulse rewards in the DM parallel version turns out to be unacceptable, we will have to implement possibility 2.

To accelerate the output of standard result measures as there are (1) the expected probability for a place to contain at least one token, (2) the mean number of tokens in a place, (3) the throughput of a transition and (4) the expected probability of a transition being enabled, these are provided in predefined result measure functions which can sum up all the relevant data in one sweep without the need of calling reward functions at all.

4.1 Distributed-Memory Parallelization

For a distributed state space with the relevant values $\pi_i(\tau)$ and $\theta_i(\tau)$ stored with the corresponding state $\boldsymbol{m}_i$ rate rewards may be computed locally without any communication. Just in the end the local sums have to be sent to the

master process which sums them up and writes the output. The only difficulty arises from the characterizing function `enabled()` which cannot be evaluated on CTMC level just regarding the incoming arcs of the local states. Therefore this function has to consult the GSPN and perform the same (time consuming) tests formerly done in the CTMC generation algorithm (for tokens, capacities, ...) to decide if a transition is enabled in the current marking or not.

To evaluate the multiplications in Eq. 8 and 9 for arc $a_k^{i \to j}$ non-local source state information — namely the $\pi_i(\tau)$ and $\theta_i(\tau)$ values — is needed. Therefore impulse rewards are expensive with respect to computation time. The strategy to compute them is exactly the same as described in Sec. 3.1 for the vector/matrix multiplications: either in one step or in $\sharp p$ steps using only parts of the vectors.

The reason why impulse rewards may not depend on the current marking by the use of NCF is, that the source marking may be non-local. In contrast to the probability values we consider the markings too big to be exchanged between the processors. The reorganization of the storage scheme (which is enforced by the CTMC generation) from incoming arcs to outgoing ones seems also not feasible due to communication overhead and memory balance.

4.2 Shared-Memory Parallelization

The shared-memory parallelization is as simple as the transient solution without the need of any locking variables just a few barriers synchronize the threads. Each thread p again is responsible for a subset $\mathcal{M}_p$ of the CTMC states.

In an initialization step all the incoming arcs are converted into outgoing ones as this is the ideal storage scheme to evaluate the characterizing function `enabled()` in a given marking by just looking at all of the transition labels of its outgoing arcs. This means each thread has to remove all the incoming arcs of its states and assign them to their source states possibly belonging to another thread. To avoid conflicts in the accruing outgoing arc lists this is done in $\sharp p$ steps again ($\sharp p$ number of processors) analogous to the computation of the diagonal elements of matrix $\boldsymbol{Q}$ at the beginning of the transient solution (see Sec. 3.2). The computation time for this initialization is negligible. The computation of the result measures itself is done completely independent in parallel on the different subsets of states and arcs.

5 Measurements

The distributed-memory (DM) algorithms were implemented using PVM and are measured on a Cluster of 16 PCs with Pentium II 300 MHz processors and 256 MB main memory connected by a Fast Ethernet (100 Mbit/sec).[4] The shared-memory (SM) algorithms are built on top of the standardized POSIX pthread library. The SM measurements were done on a Convex SPP1600 machine with

[4] We thank the Paderborn Center for Parallel Computing for enabling time measurements without influence from other processes.

895 MB main memory and eight 120 MHz HP PA-RISC processors. Memory is accessed via a crossbar. The sequential runs comparing ℙANDA with SPNP were done on a 166 MHz Sun UltraSparc-1 machine with 256 MB main memory as we have only access to the SPNP binaries for the Sun Solaris operating system.

architecture	Sun UltraSparc			Pentium II		Convex SPP
main memory	256 MB			8×256 MB	16×256 MB	895 MB
tool	SPNP	ℙANDA		DM		SM
optimization	no	no	yes	yes		yes
♯ processors	1			8	16	8
largest solvable CTMC						
states	530 000	585 000		2.5 mio	4.6 mio	2 mio
arcs	4.9 mio	5.5 mio		24 mio	44 mio	19 mio
CTMC gen. time	?	55 min	17 min	53 min	93 min	27 min
total time	74 min	64 min	24 min	67 min	120 min	103 min

Fig. 6. CTMC sizes and their computation times (mio = million, min = minutes).

All the measurements were made with a GSPN which models a multiprocessor system with failures and repairs taken from [15]. The standard result measures for one time point $\tau = 50$ for each place and transition are computed. The model can be scaled by increasing the number of tokens representing the processors in the system, the number of busses was fixed to 3. As usual the bigger the model gets the coarser is the scaling that results by adding one more token.

The single processor versions are really sequential without communication and synchronization and without computational overhead caused by the parallelization.

Fig. 6 shows the number of states and arcs for the largest CTMCs that can be generated and solved on 1, 8 and 16 processors of different architecture. Increasing the model is either not possible because the next bigger CTMC does not fit into memory (this is the case for the SPNP and DM runs) or solution times degrade ($> 4h$) due to swapping effects (ℙANDA and SM). The CTMC generation time and the total time for analysis and result measure computation are given. For simplicity in measuring and comparison the number of iterations of the transient solution was artificially limited to 20.

The data of the first two columns gives a notion how our implementation performs compared to an established sequential tool, namely SPNP 5.1. Since we only had SPNP binaries at our disposal that were compiled without a compiler optimization flag, the times of the sequential runs of ℙANDA are given with and without compiler optimization. If only CTMC generation was regarded, SPNP would able to create larger state spaces than ℙANDA since the memory for the numerical solver is allocated after the generation is finished thus running out

of memory not before the numerical solution starts. ℙANDA on the other hand prepares the data structures already during generation for containing some of the values of the numerical solution. This way the space of the search data structure may be reused which results in slightly bigger solvable models.

The parallel runs in the DM columns show that the model size from 1 to 8 processors scales by a factor of about 4.3. Beside algorithmic reasons this is due to the data structures of the DM implementation being much more wasteful than necessary at the moment. Therefore the model size from 8 to 16 processors scales better namely by 1.83. This shows that the algorithm itself works well. The loss compared to an optimal scaling factor of 2.0 can be explained by the number of message buffers and the overhead in balancing the state space which increase with a growing number of processors. As we will see later, some parts do not gain good speedups which results in an increase of solution time.

To our surprise it was also possible to use the one-step multiplication for the two models given in the DM columns. This is because swap space was accidentally big enough. But for these model sizes the multiple step solution is already 10% faster: the swapping effects compensated for the communication overhead.

The last column shows the computation times for the biggest model solvable on the Convex SPP1600. The size scales optimally as no memory is lost for send and receive buffers. We could also generate the CTMC with 4.6 million states on this machine with "only" 895 MB of shared memory. Generation time was only 97 minutes using the SM approach and 8 processors. But during the numerical solution swapping occurred, so we interrupted the ongoing solution after 6 hours. This shows that the generation phase in contrast to the solution phase can use swap space efficiently.

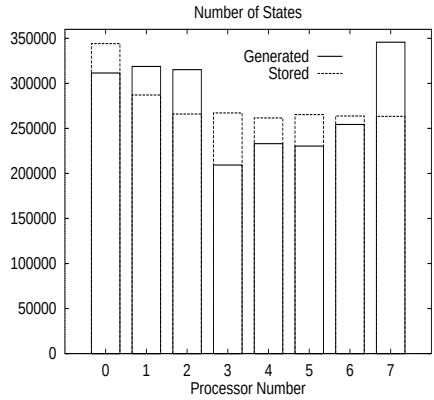


Fig. 7. Distribution of states onto processors (distributed-memory).

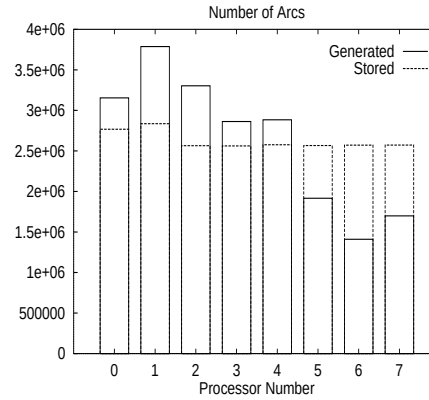


Fig. 8. Distribution of arcs onto processors (distributed-memory).

Fig. 7 shows the number of states generated on a given processor and stored on it finally for one DM run for a CTMC with 2.2 mio states and 21 mio arcs on

8 processors. Fig. 8 shows the same information as Fig. 7 for the arcs (note the different scale). The number of generated states and arcs differ very much across the processors but load balancing equalizes them resulting in slightly varying numbers of stored arcs and more varying numbers of stored states. As there are ten times more arcs than states the arcs are decisive which leads to a maximum difference of stored bytes of 10% which was the tolerance parameter we defined for load balancing in this run (see Page 153).

In Fig. 9 the computation times of the DM algorithms for the total analysis process: CTMC generation, uniformization and standard result computation are given for a small state space of 260 000 states and 2.4 mio arcs where no swapping occurs on one processor. The only part that gains a considerable speedup is the CTMC generation. This proves that the dynamic load balancing algorithm works efficiently. The share of load balancing in computation time never exceeded 20%.

As the model is small using more than 6 processors leads to an increase in solution time. The one-step multiplication is faster than the multiple step multiplication in the transient solution since only one sweep through all the arcs has to be performed thus avoiding idle times caused by an unequal distribution of operations in one of the multiple steps. The generally poor parallel performance

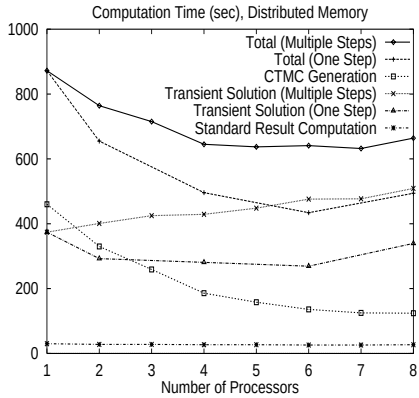


Fig. 9. Computation times for the DM algorithms for a CTMC with 260 000 states.

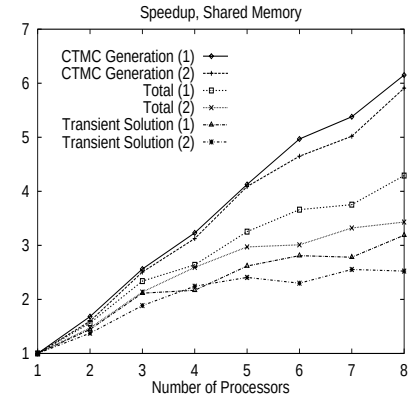


Fig. 10. Speedups for the SM algorithms for two different CTMCs.

of the uniformization is not astonishing as in each iteration a vector of size $|\mathcal{M}|$ has to be exchanged but only few arithmetic operations are done. Therefore we do not think that any numerical solvers starting from this scattered state space can gain good speedups, so we are content that times do not degrade. As expected, result computation is the fastest part of the analysis also in the parallel version.

Fig. 10 shows the speedups for the SM algorithms for two CTMCs with (1) 1.6 mio states and 14 mio arcs and (2) 700 000 states and 6.6 mio arcs.

The CTMC generation speedup is linear and the better the bigger the model. For the transient solution it is less good but acceptable. This is not due to synchronization (as we described in Sec. 3.2 there is hardly one) but to memory accesses made for both the destination and source state of each arc. This means memory accesses are more scattered and caches have to be updated more often for the uniformization. This is also the reason why swapping mainly affects the numerical solution.

We investigated result computation times also for user defined rate and impulse rewards but experienced that several very complex result measures have to be specified to find expression in a remarkable increase of computation times.

6 Conclusion

Due to our distributed-memory algorithms we can now analyze models on local area networks of sizes which are beyond the capacity of single workstations. We showed that dynamic load balancing distributes the state space equally onto the processors. This is important to guarantee scalability for any arbitrary model.

To our experience the shared-memory architecture proves to be more suitable for the GSPN analysis than the distributed-memory architecture with respect to speedup, memory utilization and implementation effort. The main argument against shared-memory machines is their availability. But we think parallel high-performance computers will be widely accessible in compute centers with the Internet getting more and more popular. As even in the workstation sector two-processor machines are common nowadays and as every important operating system comes up with a standardized thread library, such acceleration approaches will get more and more significant.

In the future we will also have to focus on more memory efficient internal data structures as up to now we simply have been more interested in the algorithmic aspects. The possibility to dynamically assign new processors to a problem if it runs out of memory will be included. This is only possible because of the flexible load balancing routine and seems very promising as too many processors increase computation times. With respect to the transient solution we want to look for memory organizations which allow a more efficient use of swap spaces.

References

1. S. Allmaier and S. Dalibor. PANDA — Petri net ANalysis and Design Assistant. In *Tools Descriptions, 9th int. Conference on Modeling Techniques and Tools for Computer Performance Evaluation (Tools'97)*, St. Malo, France, 1997.
2. S. Allmaier and G. Horton. Parallel shared-memory state-space exploration in stochastic modeling. In G. Bilardi et al, editor, *Solving Irregularly Structured Problems in Parallel*. Springer, LNCS 1253, 1997.
3. G. Balbo. On the success of stochastic Petri nets. In *Proc. IEEE Int. Conf. Petri Nets and Perf. Models (PNPM '95)*, pages 2–9, Durham, NC, 1995.

4. S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In *Proc. IEEE Int. Conf. on Appl. and Theory of Petri nets*, Torino, Italy, 1995.
5. G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, and K.S. Trivedi. Automated generation and analysis of Markov reward models using stochastic reward nets. In C. Meyer and R.J. Plemmons, editors, *Linear Algebra, Markov Chains and Queueing Models*. Springer, 1992. IMA Vol. in Math. and Appl., vol. 48.
6. G. Ciardo, J. Gluckman, and D. Nicol. Distributed state-space generation of discrete-state stochastic models. *ORSA J. Comp.* To appear.
7. G. Ciardo and A.S. Miner. Storage alternatives for large structured state spaces. In R. Marie et al, editor, *Computer Perf. Eval.* Springer, LNCS 1245, 1997.
8. G. Ciardo, J. Muppala, and K.S. Trivedi. SPNP: stochastic Petri net package. In *Proc. IEEE 3rd Int. Conf. Petri Nets and Performance Models (PNPM '89)*, pages 142–151, Kyoto, Japan, 1989.
9. D. Comer. The ubiquitous B-tree. *Computing Surveys*, 11(2):121–137, 1979.
10. Message Passing Interface Forum. MPI: a message-passing interface standard. *J. Supercomputer Appl. and High Perf. Comp.*, 8, 1994.
11. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, 1994.
12. R. German. SPNL: Processes as language-oriented building blocks of stochastic Petri nets. In R. Marie et al, editor, *Comp. Perf. Eval.* Springer, LNCS 1245, 1997.
13. D. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 1973.
14. P. Marenzoni, S. Caselli, and G. Conte. Analysis of large GSPN models: A distributed solution tool. In *Proc. IEEE Int. Conf. Petri Nets and Perf. Models (PNPM '97)*, St. Malo, France, 1997.
15. M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance models of multiprocessor systems*. MIT Press, 1986.
16. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. Wiley, 1995.
17. A.L. Reibman, R.M. Smith, and K.S. Trivedi. Markov and Markov reward model transient analysis: an overview of numerical approaches. *European J. Operations Research*, pages 257–267, 1989.
18. A.L. Reibman and K.S. Trivedi. Transient analysis of cumulative measures of Markov model behaviour. *Stochastic Models*, 5(4):683–710, 1989.
19. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, New Jersey, 1994.

SWN Nets as a Framework for the Specification and the Analysis of FT Techniques Adopted in Electric Plant Automation*

Lorenzo Capra[1], Rossano Gaeta[1], and Oliver Botti[2]

[1] Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149, Torino, Italy
{lorenzoc,rossano}@di.unito.it
[2] ENEL S.p.A., Dept. of Electrical & Automation Research
Via Volta 1, 20093, Cologno Monzese, Italy
botti@pea.enel.it

Abstract. The use of formal methods for specification and analysis of dependable systems is considered a promising opportunity to support the evaluation of critical issues since the early design phases. Stochastic Petri nets can play an important role not only for the specification of functional issues of a system, but also for the predictive evaluation of performance and dependability properties. In this paper we investigate the possibility of using Stochastic Well-formed Nets (SWN) as a framework for specifying, validating and evaluating fault tolerance mechanisms used in plant automation. A temporal redundancy technique currently adopted in several electric plants to deal with transient faults is taken as a case-study. The peculiar feature of SWNs is the capability of directly generating an aggregated state space thus allowing for efficient model analysis.

1 Introduction

The increasing complexity of automation systems, which combine high functional, real-time and fault-tolerant requirements, demands techniques and tools to support design choices and validation phases. Support is needed for the proper selection of fault tolerance (FT) mechanisms suited to obtain the required level of dependability [11],[12] and for the qualitative and quantitative analysis of system properties. The need for this kind of decision support comes not only from the traditional field of safety critical systems, but from mission critical ones (like energy production and distribution) where the economic impact of possible failures on the production process is very high. The use of formal methods for specification and analysis of dependable systems is considered a promising opportunity to face with critical issues since the early design phases.

Stochastic Petri nets can play an important role not only for the specification of functional issues of a system, but also for the predictive evaluation of performance and dependability properties. Previous applications of these techniques

* This work was supported in part by the Italian Ministry for University and Research and in part by the Esprit project TIRAN.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 168–187, 1999.
© Springer-Verlag Berlin Heidelberg 1999

(e.g., [9]) in the plant automation field show that clear, unambiguous and executable specifications, together with automatic tools supporting simulation and analysis, allow to better reason on the critical aspects, driving design choices and providing a significant contribution to verification and evaluation activities. Furthermore, they reduce the risk of system re-design due to late discovery of unfulfilled requirements.

In this paper we investigate the possibility of using a class of high-level stochastic Petri Nets known as Stochastic Well-formed Nets (SWN) [5] as a framework for specifying and deriving both qualitative and quantitative properties of FT mechanisms used in plant automation. A temporal redundancy technique based on a combination of distributed software and dedicated hardware components is taken as a case-study. Such a technique is the main component of a layered FT architecture, used in several ENEL[1] plants to deal with transient faults. A novel implementation of these FT solutions is foreseen [12],[13] to be adopted for the next generation of ENEL automation systems. The experimentation of SWN techniques carried on by the present work is aimed to provide the basis for an evaluation framework to support the development of these new FT solutions.

Among the different proposals of high level extensions of Generalised Stochastic Petri Nets (GSPN) [4], SWNs offer the possibility to *directly* generate an aggregated state space (*symbolic reachability graph* or SRG) that preserves *all* the information of the reachability set. From the stochastic point of view, it is possible to generate a *lumped Markov chain* isomorphic to the SRG [5], thus reducing the complexity of the analysis of the stochastic process underlying the SWN model as well as the discrete-event simulation [7]. The SRG construction relies on the concept of *symbolic marking*. Roughly speaking, a symbolic marking represents all such markings that can be obtained from each other by admissible objects permutations.

The paper is organised as follows: Sect. 2 describes the system under investigation; Sect. 3 gives an informal presentation of the SWN formalism and outlines the technique employed to represent deterministic timing using null constant and exponentially distributed random delays; Sect. 4 presents the specification of the temporal redundancy mechanism by SWN models. The qualitative analysis of the model is then carried out in Sect. 5 where a preliminary structural analysis is performed, and in Sect. 6 where the analysis is based on the aggregated state space directly generated from the SWN model described in Sect. 4. Sect. 7 deals with the probabilistic representation of fault processes by SWN models and an example of quantitative steady-state analysis of the system with respect to different characterisations of fault occurrence is presented in Sect. 8. Finally, Sect. 9 draws some final remarks and outlines future development of the present work.

[1] ENEL is the main Italian electricity supplier.

2 The Temporal Redundancy Technique for Electric Plant Automation

In electric plant automation the main source of faults is the electricity itself: electro-magnetic interferences may produce permanent and transient faults affecting computation, I/O and memory. Since the early 80s, ENEL defined a FT architecture based on software and dedicated hardware, initially applied to the High Voltage Substation Controller [8], then widely used for different plants and today under a renewing process [11],[12]. A number of typical mechanisms for error detection, isolation and recovery are provided within the architecture whose structure is the following: the *application layer*, which concerns command, control and monitoring of the plant; the *support software* layer (SSW), which implements the FT mechanisms providing the application layer with a sort of safe virtual machine; the *hardware layer*, which is composed of a central cabinet (on which application and SSW run), I/O units and dedicated devices.

The core of the FT strategy is represented by the *stable memory* (SM), a storing device exploiting temporal and spatial redundancy, error detection and correction techniques, to protect against transient faults affecting input, elaboration and memory and against permanent faults affecting memory. SM is used by SSW to store and stabilise the application internal state. The interaction between SSW and the application is realised through a portion of unprotected RAM, called *exchange RAM* (ExRAM), so decoupling the two layers. The interaction of SSW with the SM dedicated boards is realised by means of calls to the SM read/write routines. The portion of SSW in charge of managing SM will be denoted by SMSW.

The application is seen as a hierarchical organisation of Mealy and Moore automata, the latter representing the interface with the plant. The application state, stored in *state variables*, is represented by the state of all its automata. The application is cyclic. Its input at each cycle is represented by its *current* state, fetched from the SM, and input signals, sampled from the field. At the end of each computation cycle the application produces a new state (called the *future* state) and commands to be sent to the field.

SM is physically implemented by two groups of specialised boards, each controlled by a microprocessor, containing the current (stable) and the future (to be stabilised) state of the application, respectively. Analogously, ExRAM is partitioned in two areas (ExRAM_c, ExRAM_f). Each SM group is internally partitioned into a read-only (RO) address space and a write/read (WR) address space, accessible by means of read/write operations, respectively.

The temporal redundancy implemented by the SM requires the application to be repeatedly executed a given number of times with the same input and state before validating its future state by a voting mechanism. Re-execution of both application and SSW is periodically forced by a cyclic hardware restart. When the computed future state is confirmed for a number of consecutive cycles, the SM groups switch, i.e., the computed future state becomes the current state.

The WR space of a group is identified with the first position of a circular buffer, having three positions. A write operation, which follows an application

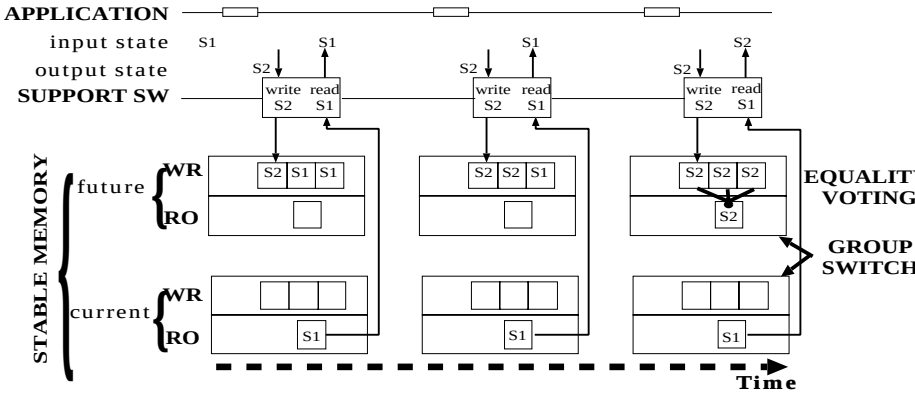


Fig. 1. SM schema and application cycle

```
phase 1: Write(ExRam_f,f_group);
phase 2: Read(&conf1,g1); Read(&conf2,g2);
         if (conf1.flag == conf2.flag) then
           c_group=g1; ExRam_c.flag=conf1.flag; ExRam_f.flag=conf2.flag;
         else
           c_group=g2; ExRam_c.flag=conf2.flag; ExRam_f.flag=conf1.flag;
         endif
phase 3: Read(&ExRam_c,c_group);
phase 4: Read(&conf,f_group);
         if (conf.sv==ExRam_f.sv) then
           ExRam_f.flag=not(ExRam_f.flag);
         endif
phase 5: Start_appl(ExRam_c,&ExRam_f);
```

Fig. 2. Pseudo-code of SMSW

cycle, causes a buffer shift and an equality voting is applied on the buffer positions. Positive voting results in copying the contents of the WR space into the RO space. The SM schema and an application cycle are depicted in Fig. 1. The structure of the SMSW is illustrated by the pseudo-code of Fig. 2; it represents the sequence of operations executed on the central cabinet unit during a cycle. The adopted "two-phases" switch mechanism is pointed out, which relies on complementing the state flag of either SM groups, once the computed future state has been confirmed. The syntax we adopt for a write is *Write(State_conf,Group)*, where *State_conf* denotes an area of the ExRam containing a state configuration. Similarly, the syntax of a read is *Read(&State_conf,Group)*. Suffixes and prefixes *c* and *f* are used instead of "current" and "future", respectively.

During phase 1 the configuration stored in ExRam_f is saved in the future group. During phase 2 state flags of both groups are read, and copied into the ExRam; if they are equal group $g1$ is set as current, otherwise group $g2$ is set as current. In phase 3 the current state is read from the current group and

copied into ExRAM_c and in phase 4 the state read from the future group is compared with the state stored in ExRAM_f (the previous output) disregarding the state flag. If they are equal, the state flag of the future group is complemented in ExRAM. During last phase the application starts, taking its input from ExRam_c, and saving the output into ExRam_f.

The first goal of this paper is to give a formal specification of the application cycle; both the the stable memory hardware and the its interaction with the SMSW will be modeled. We assume that the SM hardware devices exploit spatial redundancy, error detection and correction techniques, thus these features will not be considered.

3 Stochastic Well-formed Nets

Stochastic Well-formed Nets (SWN) [5] are a high-level extension of Generalised Stochastic Petri Nets (GSPN) [4]. As in GSPNs, there are two kinds of transitions: *timed* transitions, which are associated with a random firing delay with a negative exponential probability distribution function, and *immediate* transitions, which fire in zero time and with priority over timed ones. We only recall that as a result of this time representation, the reduced Reachability Graph of a GSPN (SWN) (obtained by deleting the *vanishing* markings, i.e. the markings where at least one immediate transition is enabled), called *tangible* RG, is isomorphic to a Continuous Time Markov Chain (CTMC). We refer the reader to [4] for a detailed explanation of the timing semantics of GSPNs.

The definition of a particular syntax to specify colour domains, arc functions and predicates forms the basis for an *automatic* detection of the model symmetries, and of their exploitation in the model solution, through the concept of *symbolic marking*. In the sequel of this section a semi-formal description of the SWN formalism is given, referring to [5] for a formal presentation.

3.1 The SWN Formalism

The entities of a system under study are called *objects*; they are collected in pairwise disjoint *basic colour classes*, that may be *ordered* if such a relation exists among the objects of the class. Colour classes may be partitioned in *static subclasses*; this partitioning is useful when it is necessary to model different qualitative and/or quantitative behaviours of objects of the same nature. We use the term *unitary* to denote a static subclasses containing only one object.

Each place P is associated with a *colour domain*, denoted as $C(\mathrm{P})$, and defined as the Cartesian product $C_1 \times \ldots \times C_n$ of (possibly repeated) basic colour classes C_i. Places associated with an empty colour domain correspond to places of a non coloured Petri Net.

Each arc, connecting place P and transition t, is labeled by *colour functions*, denoted by I, O, H, depending on the arc type (input, output, inhibition). They are defined as weighted formal sums of *basic function* tuples, $\langle f_1, \ldots, f_k \rangle$, whose arity is equal to $C(\mathrm{P}) = C_1 \times \ldots \times C_k$. Each f_i may be one of the following types:

$SC_{i(sbc)}$: Synchronisation (constant) function: it returns all objects of $C_{i(sbc)}$, where $C_{i(sbc)}$ denotes a static subclass of C_i. If $C_{i(sbc)}$ is not specified then S returns all objects of colour class C_i.

x : Identity function (hereafter called *variable*): when bound to $c \in C_i$ it returns the object itself (i.e. $x(c) = c$). $f, y, z, \ldots$ are also used as variables to denote other colour components.

$!x$: Successor function: it is defined only on ordered basic colour classes. It returns the successor of the object returned by x. The ordering is circular, i.e. $!^{|C_i|}x = x$.

Let $Var(t)$ denote the set of variables appearing in colour functions labeling the arcs from/to transition t. For a given $x \in Var(t)$, $C(x)$ denotes the colour class whose objects are bound to x.

A predicate may be associated with transition t (denoted as $Pr(t)$); it is obtained by combining basic predicates by means of the logical operators $\vee$ and $\wedge$. A basic predicate may be one of the following types: $x = y$, $x \neq y$, $x = !y$, $x \neq !y$, $x \in C(x)_{sbc}$, $x \notin C(x)_{sbc}$, where $C(x) = C(y)$ and $C(x)_{sbc}$ denotes a static subclass of $C(x)$.

An ordinary marking is a function mapping each place P into a multi-set (denoted by a weighted sum) $M(\mathrm{P})$ over $C(\mathrm{P})$. [2] The *initial marking* is denoted as M_0. Tokens will be denoted by tuples $\langle c_1, .., c_n \rangle$ where $c_i \in C_i$, $i = 1, \ldots n$.

A binding b of the variables of $Var(t)$ to objects of the associated colour classes is called a *colour instance* (c.i.) of t (also denoted by (t, b)). Symbols $I(\mathrm{P}, t)\langle b \rangle$, $O(\mathrm{P}, t)\langle b \rangle$, and $H(\mathrm{P}, t)\langle b \rangle$ denote, for a given c.i. (t, b), and for a given input, output, or inhibitor arc connecting the place P to t, the evaluation of the corresponding colour function (producing a multi-set over $C(\mathrm{P})$). For a given marking M, a c.i. (t, b) with the following properties:

- the predicate on b holds;
- for each input place P, $I(\mathrm{P}, t)\langle b \rangle \leq M(\mathrm{P})$;
- for each inhibitor place P, $H(\mathrm{P}, t)\langle b \rangle > M(\mathrm{P})$

is said to *have concession* in M and will be called *occurrence mode* (o.m.). An o.m. is *enabled* in $M \Leftrightarrow$ it does not exists a higher priority transition that has o.m. in M. Let $E(M)$ denote the set of o.m. enabled in M. An o.m. $(t, b) \in E(M)$ may fire, producing a new marking M' (denoted as $M[(t, b)\rangle M'$) such that for each place P, $M'(\mathrm{P}) = M(\mathrm{P}) - I(\mathrm{P}, t)\langle b \rangle + O(\mathrm{P}, t)\langle b \rangle$. A firing (or occurrence) sequence is a sequence of markings and o.m. $M_1[e_1\rangle M_2[e_2\rangle M_3 \ldots M_n[e_n\rangle M_{n+1}$. The set of all markings reachable form M is denoted by $[M\rangle$ ($[M_0\rangle$ is called *reachability set*).

A peculiar feature of SWN is the definition of the so called *symbolic marking*; each symbolic marking is an equivalence class of ordinary markings, up to permutations on basic colour classes preserving the partition into static subclasses. Symbolic markings are expressed in terms of *dynamic subclasses*, which identify

[2] Hereafter the $>, \leq$ relations, the$+, -$ operations, and cardinality will be implicitly extended to weighted sums, considering the respective coefficients.

possible dynamic partitions of a colour classes. A symbolic firing rule is defined, which allows the SRG of a SWN to be directly constructed from the symbolic initial marking. The main advantage is that it is possible to *directly* and *automatically* generate an aggregated state space (called *symbolic reachability graph* or SRG) that preserves *all* the information of the reachability set. From the stochastic point of view, the complexity of the solution of the stochastic process underlying the SWN model is greatly reduced, as discussed in [5].

3.2 Graphical Notation

We adopt the following conventions for the SWN models that will be presented in the paper. Place names will be written in small capitals while transition names will be written in italic. The top part of each figure depicting a SWN model will contain a list of places colour domain and transition predicates; the places that do not have an associated colour domain and transitions without predicate will not be listed. The notation $\pi : h$ drawn near an immediate transition t will be used to highlight its priority level and will be used only when $h \geq 2$. The steady-state throughput of t will be denoted as $X(t)$.

If there exist one input and one output arc connecting place P and transition t such that $I(\mathrm{P}, t) = O(\mathrm{P}, t) = k$, we denote them as *test arc* and will draw a single, double arrowed arc with multiplicity equal to k. Similarly, if there exist one input and one inhibitor arc connecting P and t such that $H(\mathrm{P}, t) = I(\mathrm{P}, t)+1$, we denote them as *flush arc* and will draw a single, double arrowed, dotted arc with multiplicity equal to k. Multiplicities will be drawn only if $k \geq 2$.

4 The SWN Model

The starting point of this work was the existing description of SSW, based on a mix of natural language and high-level pseudo-code. Focusing on SM, our first aim was to provide a formal specification, by means of a SWN model, of the operations executed by SSW on SM boards during a cycle.

To cope with the layered design of the FT technique, SWNs have been integrated with a simple, modular approach inspired by the Process/Resource-Box (P/R-B) [10] modeling technique. The P/R-B allows a step-wise representation of both software processes and software/hardware resources. Moreover, in terms of P/R-B, the mapping of a process over a resource is obtained by applying well-known net operators to box interfaces (in our case, transition fusion).

We present SM as an available composite resource. The process accessing this resource is represented by the composition of SMSW and an abstract application. The complete model, here omitted for space reasons, has been obtained by composing the Resource-box of the SM (given in Sect. 4.1) with the Process-boxes of SMSW and of the abstract application (given in Sect. 4.2).

In each sub-model, variables denoted by capital letters occur in arc inscriptions of transitions belonging to the box interface, representing formal parameters. The composition of submodels by transition superposition requires a match-

ing (colour respecting) of formal parameters appearing in arc inscriptions of superposed transitions (Sect. 4.3).

The colour classes we defined for the SWN model are: $GROUP = g1 \cup g2$, is an ordered class partitioned into two unitary static subclasses which identify the SM groups. Colour class $SV = conf \cup s0$ is a non-ordered class partitioned into two static subclasses which are the basis of a very effective, "two-state" representation of the mechanism. The objects in $conf$ denote different states of the application, while $s0$ is used as "refresh" value. Class $FLAG = \{true, false\}$ is used to represent the boolean values assigned to flags. Class $ROLE = cr \cup fr$ is non-ordered and partitioned into two unitary static subclasses that represent the roles alternatively played by the SM groups ("current" and "future") and the partition of the ExRAM.

Most of the activities of the system we investigate are reasonably assumed to have a constant duration, e.g., reads and writes on memories, while application execution times and fault periods may better be modeled using random delays. Constant delays, in the system we consider, are multiples of a time unit (the access time to the stable memory is equal to 20ms) therefore, we adopt a modeling approach based on the use of only one exponential transition that models a system clock, and an arbitrary number of immediate transitions that represent the system operations. More details on this approach, adopted in context of ATM communication networks modeling, can be found, for instance, in [1],[2],[3].

The firing of the clock transition (that will be named $Clock$) deposits one token into a non-coloured place (named TIME) thus modeling the progression of system time. The number of tokens in place TIME represents the time units elapsed since the starting of an application cycle (since in our model a time unit is taken to be equal to 20ms, $M(\text{TIME}) = d \Rightarrow d \cdot 20$ milliseconds have elapsed). The number of tokens in place TIME does not grow unboundedly. Its content is emptied by the firing of any transition representing the end of a cycle, corresponding to phase 5 of the pseudo-code sequence. This mechanism is modeled by connecting these transitions to place TIME through a flush arc with the appropriate multiplicity.

Transitions that represent phases in the SMSW pseudo-code are allowed to fire only at well defined time instants, in accordance to the represented phase position in the pseudo-code sequence. To achieve this, some of the model transitions have test arc connections with place TIME whose multiplicity reflects the time units that must elapse before the corresponding activity may take place, since the beginning of a cycle.

4.1 The Model of the Stable Memory Hardware

The model of the hardware of the SM is depicted in Fig. 3, Both SM read/write and voting operations are represented as atomic by means of single transitions. The time taken by the voting procedure, reflecting its actual complexity, is expressed by the multiplicity of the test arc connection between place TIME and transition $Vote$. For the remaining transitions of Fig. 3, the multiplicity of the test connection with the place TIME is left unexpressed. In fact they represent

$C(\text{WR}) = GROUP \times SV \times FLAG \times SV \times FLAG \times SV \times FLAG$ $\Pr(Vote) = [f \neq f1 \vee y \neq x]$
$C(\text{RO}) = GROUP \times SV \times FLAG$
$C(\text{FS}) = GROUP$

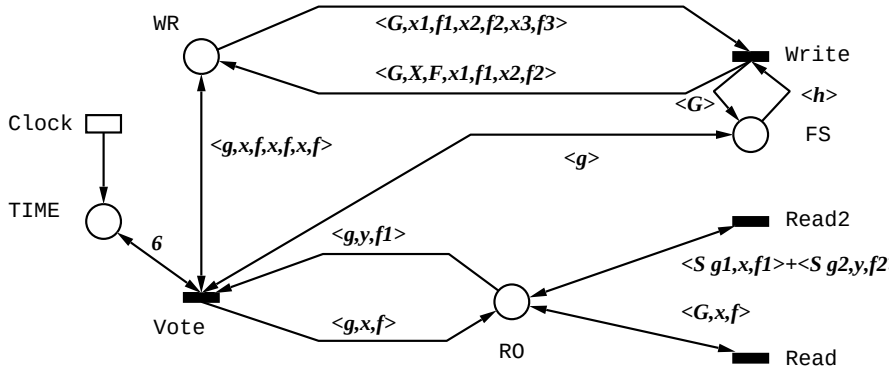


Fig. 3. Stable Memory hardware model

I/O routines called by SMSW, and they will be superposed with corresponding transitions of its Process-box.

Place WR models the WR space of a SM group, identified with the circular buffer. A pair $SV \times FLAG$ of the place colour domain represents a state configuration, which contains a flag (called state flag) used for switching the system. Place RO models the RO space of a SM group while place FS identifies the group on which the voting operation is executed.

Transition $Write$ represents a write operation on the first position of the WR circular buffer, and the buffer shift. The formal parameter G identifies the group on which the operation is executed, while the pair of formal parameters (X, F) represents the state configuration that is saved. Transition $Read$ models a read operation from the RO address space of the group G. The read configuration is represented by the pair of variables (x, f). Transition $Read2$ represents a sequence of read operations from the RO spaces of both groups. The read configurations are represented by the pairs of variables $(x, f1)$, and $(y, f2)$. It has been introduced to keep as simple as possible the composition with the SMSW model. Transition $Vote$ models a successful voting operation in a SM group (identified by g). The firing of this transition copies the configuration (x, f) stored in the RW space to the RO space.

4.2 The Model of the Stable Memory Support Software

The model of the SMSW is depicted in Fig. 4. The cyclic structure of the net represents the periodic restart mechanism. The system timing, based on counting not maskable interrupts periodically generated by a dedicated board, is simulated by the adopted clock-based approach.

The set {SAVEFS, IDENTIFYCS, COPYCS, COMPAREFS, BEGINAPPL, ENDAPPL} (denoted as CF) represents the control flow of the above sequence. The token traversing them, referred to by the variable g, denotes the "current" SM group. Place EXRAMF represents the copy of the SM state flags in ExRAM. Place EXRAMSV represents the copy of the state variables in ExRAM. We use the auxiliary place CKAPP to model the duration of the application.

Transitions are grouped into sets and each set corresponds to a phase of the pseudo-code sequence. Occurrence modes of different transitions in a set are in mutual exclusion. Transition $Writefs$ represents phase 1 of the pseudo-code sequence. Arc inscription g is related to parameter c_group while arc inscriptions x, f are in correspondence to $ExRam_f$. Transitions $Idecurr1$ and $Idecurr2$ describe phase 2 of the pseudo-code sequence. Arc inscription $F1$ corresponds to

$C(\text{ENDAPPL}) = GROUP$
$C(\text{BEGINAPPL}) = GROUP$
$C(\text{SAVEFS}) = GROUP$
$C(\text{IDENTIFYCS}) = GROUP$
$C(\text{COPYCS}) = GROUP$
$C(\text{COMPAREFS}) = GROUP$
$C(\text{EXRAMF}) = ROLE \times FLAG$
$C(\text{EXRAMSV}) = ROLE \times SV$

$\Pr(Readfsnoc)=[X \neq y]$
$\Pr(Readfsyesc)=[X = y \wedge f2 \neq f3]$
$\Pr(Idecurr1)=[F1 = F2]$
$\Pr(Idecurr2)=[F1 \neq F2]$
$\Pr(Application)=[x \in conf \wedge z \in conf \wedge x \neq z]$

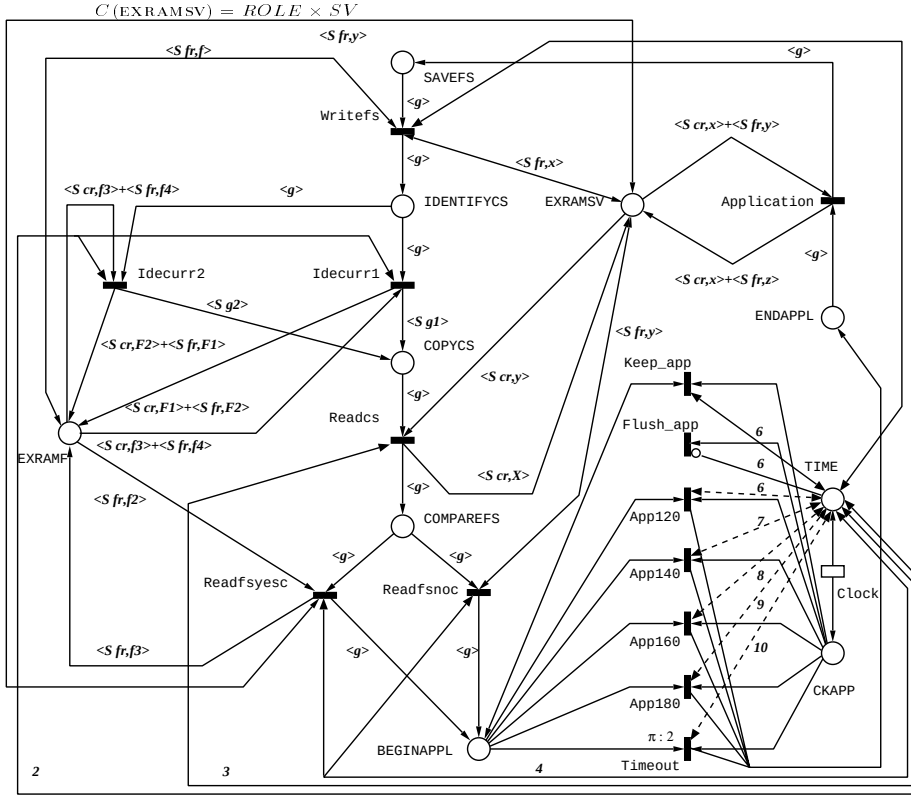


Fig. 4. SMSW and application model

$conf1.flag$ and $F2$ corresponds to $conf2.flag$. Transition *Readcs* models phase 3 of the pseudo-code sequence. Arc inscription g is related to c_group and X corresponds to $ExRam_c$. Transitions $Readfsyesc$ and $Readfsnoc$ describe phase 4. Arc inscription g corresponds to parameter c_group, X corresponds to $conf.sv$ and y is related to $ExRam_f.sv$. The following set models the duration of the application: $\{Keep_app,\ Flush_app, App120, App140, App160, App180, Timeout\}$. Transition *Application* models a correct computation, which is by convention represented changing the input state from x to z, as specified by the associated predicate (z is fixed, since $\mid conf \mid = 2$); it represents also the restart of the sequence.

In the following, we use *slc* to denote any transition sequence corresponding to a cycle, i.e. *slc* is the set family:

$$\{Writefs, (Idecurr1, Idecurr2), Readcs, (Readfsyesc, Readfsnoc),\\ (App120, App140, App160, App180, Timeout), Application\}$$

where alternatives are delimited by round brackets. The set of all transitions appearing in *slc* will be denoted by SLC.

4.3 The Sub-models Composition

The complete model is obtained by superposing the following transition pairs, where the first transition refers to the Fig. 3, while the second one refers to the Fig. 4. Substitution of formal parameters with actual functions is listed between square brackets:

$$\begin{array}{ll} Read \times Readcs & [G : g, X : x] \\ Read \times Readfsyesc & [G : !g, X : x] \\ Read \times Readfsnoc & [G : !g, X : x] \\ Read2 \times Idecurr1 & [F1 : f1, F2 : f2] \\ Read2 \times Idecurr2 & [F1 : f1, F2 : f2] \\ Write \times WriteSV & [G : !g, X : x, F : f] \end{array}$$

Notice that !g denotes the "future" group, $GROUP$ being an ordered class of two elements. We assume that transitions resulting from the composition maintain the names of Fig. 4.

5 Model Validation by Structural Analysis

Structural analysis is a useful approach for a preliminary model validation. Even if less complete than state space inspection, it proves to be necessary when state space dimensions are so large that its inspection is unfeasible. In this context, a simple structural analysis is carried out to detect basic model properties. Such properties are maintained when a fault representation is introduced (Sect. 7), which makes impossible the SRG inspection, while preserving the original system structure. More complex qualitative properties of the represented mechanism will be derived by inspection of the SRG of the model without fault representation.

Let $\Pi_i : Bag(C_1 \times \ldots \times C_k) \rightarrow Bag(C_i)$, $1 \leq i \leq k$, be the bag extension of the projection on the i-th class. For each $M \in [M_0\rangle$, the following relations hold (we assume that relations (1)-(4) hold in the initial marking):

$$(\sum_{\mathrm{P}\in CF} \mid M(\mathrm{P}) \mid) = 1 \quad (1)$$
$$\Pi_1(M(\mathrm{RO})) = \Pi_1(M(\mathrm{WR})) = 1 \cdot g1 + 1 \cdot g2 \quad (2)$$
$$\Pi_1(M(\mathrm{EXRAMF})) = \Pi_1(M(\mathrm{EXRAMSV})) = 1 \cdot cr + 1 \cdot fr \quad (3)$$
$$\mid M(\mathrm{FS}) \mid = 1 \quad (4)$$
$$\forall t, \exists !(t,b) : \forall \mathrm{P}, I(\mathrm{P},t)\langle b\rangle \leq M(\mathrm{P}) \quad (5)$$
$$\forall\ (t,b),(t',b'), t \neq t', t, t' \in SLC : (t,b) \in E(M) \Rightarrow (t',b') \notin E(M) \quad (6)$$
$$\exists M' \in [M\rangle, (t,b), t \in SLC, : (t,b) \in E(M') \quad (7)$$

Relations (1) and (4) hold since $(\sum_{\mathrm{P}\in CF} 1 \cdot \mathrm{P})$ and $1 \cdot \mathrm{FS}$ are semi-flows of the decoloured model, obtained by replacing colour functions with the corresponding cardinalities (which are colour independent), and by deleting transition predicates. Relations (2) and (3) hold since $\forall t, \forall \mathrm{P} \in \{\mathrm{RO}, \mathrm{WR}, \mathrm{EXRAMF}, \mathrm{EXRAMSV}\}$: $\Pi_1(O(\mathrm{P},t) - I(\mathrm{P},t)) = f_\emptyset$ ($f_\emptyset$ is the null constant function on multisets). Relation (1) represents the control flow of SMSW while relations (2) and (3) reflect the partition of SM and of ExRAM. Starting from relations (1)$-$(4), some further results concerning liveness and mutual exclusion (5)-(7) were obtained (proofs are omitted for lack of space). We may summarise the previous results saying that from any marking, exactly one o.m. (t,b) representing a cycle operation becomes eventually enabled. As a consequence, the net is live with respect to SLC. This reflects some features of the represented system, which is cyclic, fully deterministic, not blocking, and which evolves irrespective of faults.

Other results have been derived from the previous ones, e.g. absence of loops of o.m. of immediate transitions (trapping the model), and net boundedness for any M_0.

6 Qualitative Analysis Based on SRG

In this section we analyze the system behaviour by inspecting the SRG generated by our model. For this purpose, we assume that the transition $Vote$ in Fig. 3 is split into two transitions whose o.m. are in structural mutual exclusion: $Voteyesc$ ($Pr(Voteyesc) = [f \neq f1]$) and $Votenoc$ ($Pr(Votenoc) = [f = f1 \wedge y \neq x]$). The split is done by rewriting $Pr(Vote) = [f \neq f1 \vee y \neq x]$ in an exclusive form ($A \vee B \equiv A \vee (\bar{A} \wedge B)$). It is easy to verify that the split results in an equivalent model, i.e. $\forall M : (Vote,b) \in E(M) \Leftrightarrow (Voteyesc, b_1) \in E(M)\, exor\, (Votenoc, b_2) \in E(M)$, where b, b_1, b_2 are the same binding (up to renaming).

These transitions represent the two phases of the SM switch: $Voteyesc$ models an actual switch of the SM groups, realised by complementing the state flag of the "future" group; $Votenoc$ models the first step of the SM switch, where the computed state, confirmed for a number of consecutive cycles, is copied from the WR space to the RO space of the "future" group.

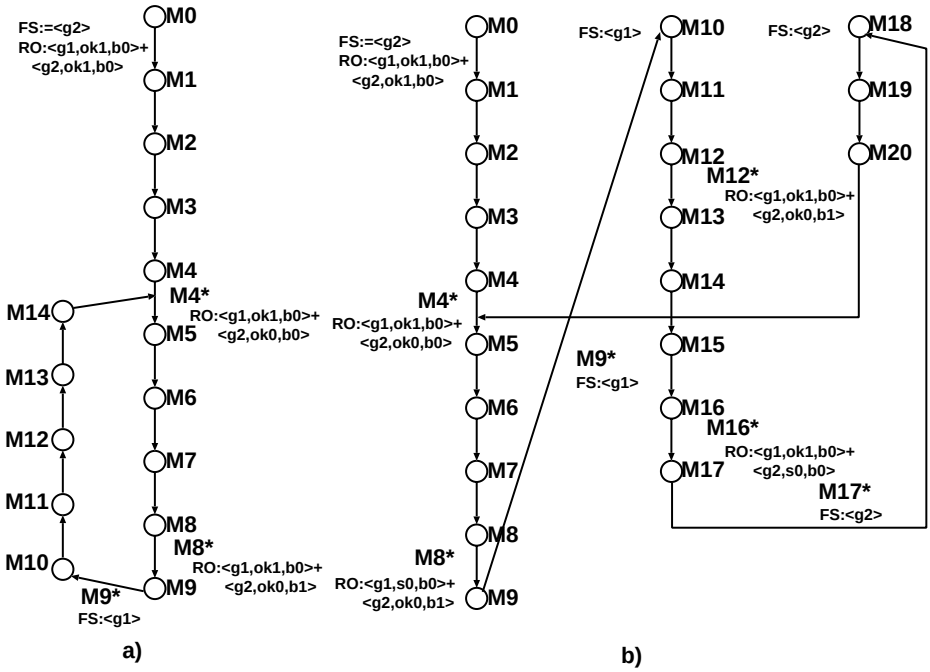


Fig. 5. SRG structures of the complete model

Let $ok0$, $ok1$ and $b0$, $b1$ denote unitary dynamic subclasses of the static subclasses $conf$ and the class $FLAG$, respectively. The symbolic initial marking of the whole model is:

$$M_0(\text{WR}) = \langle S, ok1, b0, ok1, b0, ok1, b0\rangle, M_0(\text{RO}) = \langle S, ok1, b0\rangle, M_0(\text{FS}) = \langle g2\rangle$$
$$M_0(\text{EXRAMSV}) = \langle S, ok1\rangle, M_0(\text{EXRAMF}) = \langle S, b0\rangle, M_0(\text{SAVEFS}) = \langle g1\rangle.$$

The symbolic marking M_0, which corresponds to four ($\mid conf \mid \times \mid FLAG \mid$) ordinary markings, denotes a starting configuration where the storing areas are initialised in the same way. For example, $M_0(\text{EXRAMF})$ corresponds to both $\langle S, true\rangle$ and $\langle S, false\rangle$. The SRG generated by M_0, whose cardinality is equal to 459 symbolic markings, is sketched in Fig. 5a. The following conventions are used: M_i denotes a symbolic marking such that $M_0[s_1\rangle \dots M_{i-1}[s_i\rangle M_i$, where s_k $(1 \leq k \leq i)$ is a symbolic firing sequence corresponding to a SM cycle (i.e. the support of s_k restricted to transitions in SLC has the form $\sum_t 1 \cdot t$). An arc from M_i to M_j means that $M_i[s_i\rangle M_j$ for any s_i, and represents the folding of all the possible paths from M_i to M_j, corresponding to different durations of a SM cycle. Hereafter, this will be simply denoted by $M_i \rightarrow M_j$. For the sake of readability arcs are not labeled. Finally, M_i* means that the marking of place FS and/or RO, on which we focus, has changed with respect to M_i.

Some general remarks about the SRG structures: $M_0, \dots, M_4$ are not *home markings*; there is only one *strongly connected component* (a maximal subgraph

where all nodes are reachable from each other and that will be denoted as SCC) composed by elements that are home markings. That is to say, the system, after a transient phase, reaches its steady-state behaviour (consequently, a steady-state solution exists for the MC isomorphic to SRG). Denoting with α, β arbitrary conditions on place markings, we consider this progress property : $\alpha \longmapsto \beta \Leftrightarrow$ for all possible finite or infinite occurrence sequences σ, such that $M_0[\sigma\rangle$, each state where α holds is followed by, or is itself, a state where β holds. The following properties hold, related to the marking of place RO (x, y denote any object of the class SV)

$$\langle g1, ok1, b0\rangle + \langle g2, x, b0\rangle \longmapsto \langle g1, y, b0\rangle + \langle g2, ok0, b1\rangle \quad (8)$$
$$\langle g1, x, b0\rangle + \langle g2, ok0, b1\rangle \longmapsto \langle g1, ok1, b1\rangle + \langle g2, y, b1\rangle \quad (9)$$

Properties (8) and (9) state that, starting from any configuration, the SM groups eventually switch in the correct way. The previous result may be refined by computing the number of cycles for a SM switch (NSC), which characterises the mechanism (the effect of faults on this index will be studied in Sect. 8). NSC is given by the length of the shortest path between two nodes M_i, M_j of Fig. 5 corresponding to consecutive SM switches. We have to consider only SCC, since we are interested in the steady state behaviour. Formally, let M_i, M_j be nodes such that

$$\begin{array}{l} \Pi_3(M_i(\text{RO})) \neq \Pi_3(M_j(\text{RO})), \\ \forall M_l : (M_l \rightarrow M_i) \Rightarrow (\Pi_3(M_i(\text{RO})) \neq \Pi_3(M_l(\text{RO}))) \\ \exists n \text{ markings } M_{k_1} \ldots M_{k_n}, M_i \rightarrow M_{k_1} \ldots M_{k_n} \rightarrow M_j : \\ \forall k_w (M_{k_w} \neq M_i(M_j) \wedge \Pi_3(M_i(\text{RO})) = \Pi_3(M_{k_w}(\text{RO})) \end{array}$$

then NSC is equal to $n + 1$. On the light of the two-phase SM switch, NSC is expected to be ≥ 6. Quite surprisingly, NCS turns out to be equal to five in the SRG in Fig. 5a), i.e. the distance between M_5 and M_{10}. This unexpected result is due to the simplified, two-state representation of SM: after a SM switch, the new output of the application is equal to the state stored in the RO space of the new "future" group of SM. Therefore, the state flag is early complemented, and SM switches in less than six cycles.

This points out a subtle anomaly of the mechanism: in case of a computational sequence $\ldots S_i \gg S_j \gg S_i$, where $S_i \gg S_j$ denotes a SM switch, with the computation output taken as the new input, the above situation takes place. In other words, the temporal window protected against faults lasts five cycles. If a fault occurs which affects the computation for a time period longer than five cycles, an erroneous SM switch may occur. However, because of the very high number of actual SM configurations, the probability of such an event is negligible.

To represent a more realistic system behaviour it is sufficient to modify the I/O functions of the transition $Voteyesc$, which models a SM switch, as described in Fig. 7 (where the transition is split, for the quantitative analysis). An occurrence of the modified transition $Voteyesc$ corresponds to reset the WR and RO areas of the new future group (by means of the constant $s0$), thus modelling

that after any SM switch the computed state is different from the one stored in the RO space of the new future group

The structural properties of Sect. 5 are preserved. The SRG generated after this change is sketched in Fig. 5b). The general properties about home markings and strongly connected components as well as properties (8) and (9) still hold. Instead, NSC turns out to be equal to eight, corresponding to the distance between M_{10} and M_{18}. This result can be explained in the following way: the considerable duration of the voting operation in the SM boards causes a delay of one cycle in both phases of the SM switch. Thus the data read from the future group during the fourth phase of a cycle (see the pseudo-code of Fig. 2) result from a voting carried out during the previous cycle.

Considering the arc from M_9 to M_{10} in Fig. 5b we may refine the previous result. It may be observed that a cycle is spent, since the contents of the RO space change until the switch of the SM groups actually takes place. In other words, the number of cycles for SM switch is equal to eight, but the eighth computation may be different from the previous ones, i.e. the system is protected against faults having a duration up to seven cycles.

Other interesting outcomes derived by the analysis of SRG in Fig. 5b concern a characterisation of the two phases of the SM switch. For instance, we have derived that a faulty computation is less expensive in the second phase (in terms of NSC), than in the first phase. This kind of analysis revealed to be very useful for a deep characterisation of the mechanism, allowing several hidden aspects and weak points to be pointed out. The current efforts of (re)designing a more efficient and flexible SM version (see [11]), which relies on exploiting the redundancies intrinsic in distributed architectures, take into account some of these outcome.

7 The Fault Representation

In this section the net in Fig. 4 is refined and integrated with a probabilistic representation of transient faults, based on a flexible Markov-Modulated Bernoulli Process (MMBP). The aim is to study the sensitivity of the temporal redundancy technique with respect to different characterisations of the fault process.

On one side, such an approach fits in with the first aim of this work, i.e. proving the viability of the SWN modeling technique. On the other side, it is coherent with the randomness of the main parameter characterising electromagnetic perturbations, i.e. the frequency. However, a more precise and timely description of the fault process (e.g. in terms of magnitude and duration) could be possible, acting on the parameters of the Markov Modulating model. The fault model is coupled with the system model by means of the place ERROCCUR, which signals that a transient fault occurred during a cycle, affecting the computed state. In order to represent such a wrong computation, the transition *Applerr* in Fig. 6 was introduced, which may be enabled only when place ERROCCUR is marked. Transitions *Application* and *Applerr* are mutually exclusive. The convention we adopt is that a wrong computation does not change the input state (the variable

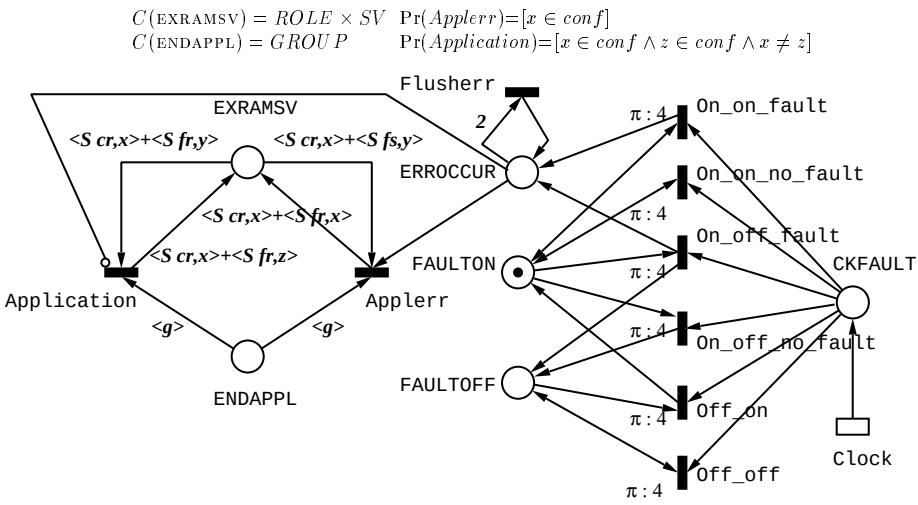


Fig. 6. Fault model with Markov-modulated fault process

z is set equal to x in arc functions from/ to place EXRAMSV). Some comments have to be made on this approach: 1) It allows the system model to be kept unchanged independently of the representation of fault occurrence; 2) representing only the final effect of transient fault is coherent with the adopted restart mechanism, which confines the effect of a fault to a single cycle, preventing at the same time the state space explosion; 3) it corresponds to the pessimistic, but realistic, assumptions that fault occurring for a number of consecutive cycles have the same effect, and that the system is fault sensitive at any instant of a cycle; 4) multiple occurrences of faults in a single cycle are not taken into account (place ERROCCUR is always marked by at most one token), as well as faults affecting the state flag (this is a reasonable assumption, because of the very large number of bits forming a state configuration).

The MMBP fault process can be either *on* (place FAULTON is marked) or *off* (place FAULTOFF is marked). If the fault process is on, the four transitions On_on_fault, $On_on_no_fault$, On_off_fault, and $On_off_no_fault$ are enabled after the firing of transition $Clock$. When the fault is in the on state an occurrence may happen with probability p. On the contrary, when the fault process is off transitions Off_on, and Off_off are enabled and no fault can occur. The firing of transition On_on_fault (On_off_fault) models the occurrence of a fault and the fault process that remains in the on state (changes to the off state). The fault occurrence on and off periods are geometrically distributed random variables, whose averages are the inverses of the probabilities $1 - P(on - on)$ and $1 - P(off - off)$. The fault *activity factor* (AF) is defined to be the ratio between the average on period duration and the sum of the average on and off period durations; with trivial algebra we get: $AF = \frac{1-P(off-off)}{[1-P(on-on)]+[1-P(off-off)]}$.

$C(\text{WR}) = GROUP \times SV \times FLAG \times SV \times FLAG \times SV \times FLAG$ $\Pr(Votenoc) = [f = f1 \wedge y \neq x]$
$C(\text{RO}) = GROUP \times SV \times FLAG$ $\Pr(Voteyescok) = [f \neq f1 \wedge x \neq z]$
$C(\text{FS}) = GROUP$ $\Pr(Voteyescko) = [f \neq f1 \wedge x = z]$

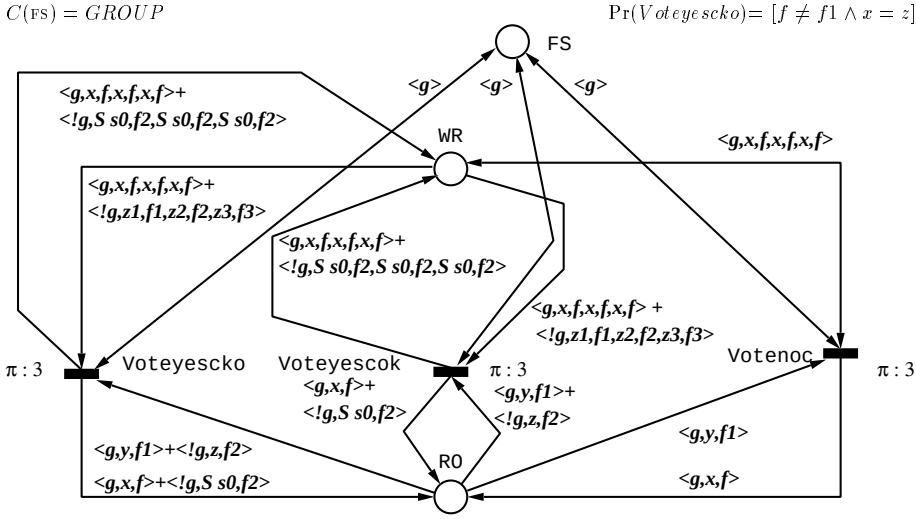


Fig. 7. Split vote model

The *average fault load factor* is defined as $\rho = p\ AF$ and the *average burst size* is $BS = p/1 - P(on - on)$. The fault occurrence interarrival times during the on periods are geometrically distributed random variables, with average $1/(1-p)$.

A further minor modification of the model can lead to the representation of Bernoulli fault process. This is achieved by deleting transitions On_off_fault, $On_off_no_fault$, Off_on, and Off_off and places FAULTON, and FAULTOFF from the model of Fig. 6.

To suitably analyze the complete model with faults, the transition $Voteyesc$ is split into two mutually exclusive transitions, $Voteyescok$, $Voteyescko$, by rewriting the corresponding predicate in an exclusive form (the resulting equivalent model is depicted in Fig. 7). These transitions represent a correct SM switch and a SM switch caused by a number of consecutive faulty computations, respectively, depending on whether the state which is being confirmed (denoted by the variable x) is different from the (old) input state (denoted by z) or not. This is in accordance with the adopted convention.

8 Quantitative Analysis: Selected Numerical Results

As an example of the numerical results that can be obtained with the SWN model, we present some curves of selected performance indices, and we comment on the complexity of the solution of the SWN model. The discussion of numerical results aims at proving the viability of the proposed SWN modeling approach, not at a complete characterisation of the steady-state performance of

the temporal redundancy technique, which is outside the scope of this paper. All numerical results were obtained with the *GreatSPN* package [6]. In the presentation of numerical results we focus on two performance indices: the mean number of cycles for SM switch (NSC), defined as $\frac{X(Writefs)}{X(Voteyescok)}$, and the erroneous SM switch probability (PER), defined as $\frac{X(Voteyescko)}{(X(Voteyescko)+X(VoteYescok))}$.

In order to validate the behaviour of the SWN model we computed the value of the performance index NSC in the case of an ideal scenario characterised by the absence of faults. The obtained value is equal to eight which is the same value derived in Sect. 6 and thus represents a lower bound for the NSC index. Numerical results are presented in the four graphs of Fig. 8, that show the NSC (top), and the PER (bottom) as functions of the fault load factor ρ. The left column presents several curves for a fixed value of the BS parameter (128) of the MMBP fault process and different values of the AF parameter, while the right column refers to a fixed value of the AF parameter (0.3) for different values of the BS parameter.

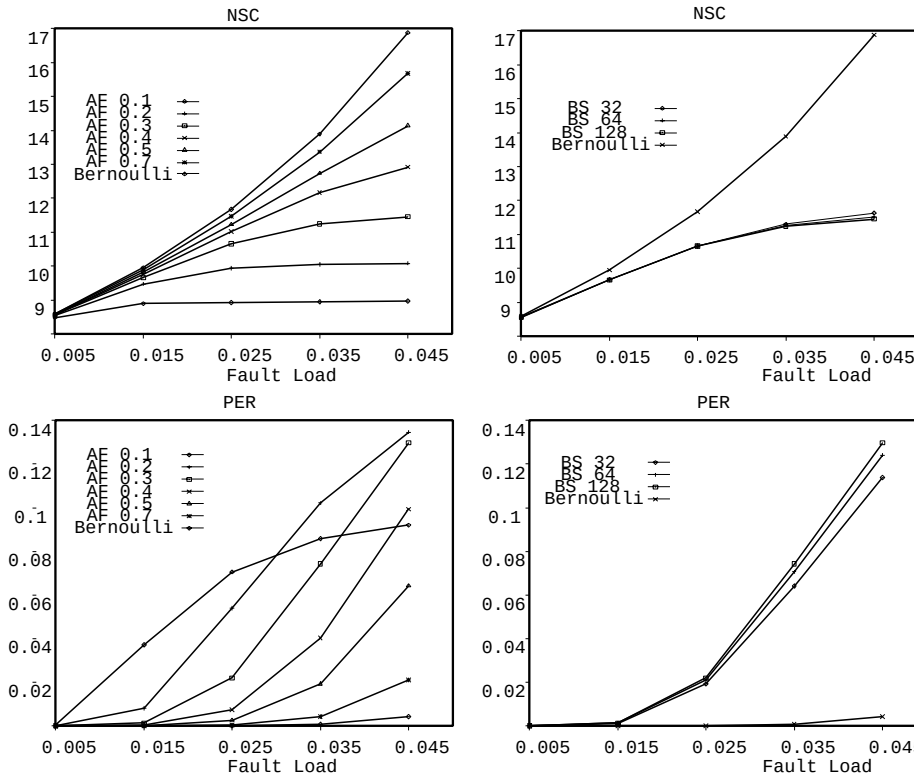


Fig. 8. Mean number of cycles for SM switch (*top*), Erroneous SM switch probability (*bottom*), as functions of the fault load factor, for either variable AF parameter (*left*) or BS (*right*)

The observation of the numerical results leads to a number of remarks. As expected, the NSC curves show increasing index values as the fault load factor increases; the same holds for the PER values. The NSC assumes higher values under the hypothesis of Bernoulli fault process, while under the assumption of MMBP fault process the lower the AF parameter the lower the NSC values. For a fixed value of the AF parameter, the NSC values are virtually insensitive to the values of the BS parameter (32, 64, and 128). This phenomenon is explained observing that low values of AF translate into long periods of correct functioning characterised by the lower bound for the NSC value that has been previously computed. Therefore, the lower the AF the longer the correctly working periods which yield lower values for NSC.

On the contrary, the PER index assumes the higher values under the assumption of MMBP fault process. This is explained observing that erroneous SM switches occur when consecutive cycles compute the same (wrong) value. Therefore, a highly correlated fault process, as the MMBP we consider in this paper, results in a higher number of consecutive faults which increase the PER values compared with the results we obtain under the assumption of a poorly correlated Bernoulli fault process.

As a final remark, the number of symbolic states generated by the model is quite low: we obtain $21,090$ and $42,178$ in case of Bernoulli fault process and MMBP fault process, respectively.

9 Conclusions, Future Works, and Exploitation

The possibility of using the class of SWN as a framework for specifying and deriving both qualitative and quantitative properties of FT mechanisms used in electric plant automation was proposed. The adopted modelling approach is based on the use of just one timed transition that models the system clock, and numerous immediate transitions that describe the system operations. A temporal redundancy technique adopted in several ENEL plants to deal with transient faults has been taken as a case-study. As a first step, the mechanism has been specified and analysed in fault-free conditions. The model has been validated and analysed by carrying out its structural analysis as well as the symbolic state space based analysis. Then, a probabilistic fault representation has been introduced and discussed, and an example of quantitative analysis has been carried out to study the sensitivity of mechanism performances with respect to different characterisations of fault occurrence. The analysis/validation through symbolic state space inspection has lead to a better understanding of the mechanism implementation, outlining some interesting unexpected behaviours. The preliminary results currently obtained from the quantitative analysis are promising and deserve further investigation taking into account steady-state and transient model analysis as well as different characterisations of the fault process. The goal is to obtain a deep characterisation of SM by estimating well known metrics like MTBF, MTTF, etc. under given environmental conditions. After this wide spectrum pilot experience, the specification and evaluation technique here proposed

has been adopted within the ESPRIT project 28620 TIRAN [13] as a support driving novel software implementations of the SM and of a complete FT framework developed for real-time embedded applications. The authors are confident that the wide exploitation and dissemination guaranteed by the TIRAN project will favour a significant penetration of SWN in the industrial environment.

Acknowledgements

The authors wish to thank Susanna Donatelli and the anonymous reviewers for their constructive comments on the original version of this paper.

References

1. L.Kant, W.H.Sanders: Loss Process Analysis of the Knockout Switch Using Stochastic Activity Networks. ICCCN 95, Las Vegas, NV, USA, September 1995
2. M.Ajmone Marsan, R.Gaeta: SWN Analysis and Simulation of Large Knockout ATM Switches. In: Desel,J., Silva,M. (eds.): ICATPN 98. Lecture Notes in Computer Science, Vol. 1420. Springer-Verlag, Berlin Heidelberg New York (1998) 326-344
3. M.Ajmone Marsan, R.Gaeta: Modeling ATM systems with GSPNs and SWNs. ACM Performance Evaluation Review, Vol. 26, n. 2, August 1998, pp. 28-37
4. M.Ajmone Marsan, G.Balbo, G.Conte: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM Transactions on Computer Systems, Vol. 2, n. 2, May 1984, pp. 93-122
5. G.Chiola, C.Dutheillet, G.Franceschinis, and S.Haddad: Stochastic well-formed coloured nets for symmetric modelling applications. IEEE Transactions on Computers, Vol. 42, n. 11, November 1993, pp.1343-1360
6. G.Chiola, G.Franceschinis, R.Gaeta, M.Ribaudo: GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. Performance Evaluation, Vol. 24, n. 1,2, November 1995, pp. 47-68
7. R.Gaeta: Efficient Discrete-Event Simulation of Colored Petri Nets. IEEE Transaction on Software Engineering, Vol. 22, n. 9, September 1996, pp. 629-639
8. R.Gargiuli, P.G.Mirandola, et al.: ENEL Approach to Computer Supervisory Remote Control of Electric Power Distribution Network. CIRED, Brighton (UK), 1981
9. O.Botti, L.Capra: A GSPN based methodology for the evaluation of concurrent applications in distributed plant automation systems. The EUROMICRO Journal of Systems Architecture, n. 42, 1996, pp. 503-530
10. O.Botti, F:De Cindio: Process and resource boxes: An integrated PN performance model for applications and architectures. Systems, Man and Cybernetics, Le Toquet, France, 1993
11. G.Deconinck, O.Botti, et al.: Stable Memory in Substation Automation: a Case Study. FTCS-28, Munich, June 1998
12. G.Deconinck, O. Botti, et al.: Reusable Software Solutions for more Fault-tolerant Industrial Embedded HPC Applications. Int. Journal Supercomputer, Vol. 13, n. 3-4, 1997
13. O.Botti, V.De Florio, G.Deconinck, et al.: TIRAN: Flexible and Portable Fault Tolerance Solutions for Cost Effective Dependable Applications. submitted for publication

Monitoring Discrete Event Systems Using Petri Net Embeddings

Christoforos N. Hadjicostis and George C. Verghese

Massachusetts Institute of Technology, Cambridge MA 02139, USA
{chadjic,verghese}@mit.edu

Abstract. In this paper we discuss a methodology for monitoring failures and other activity in discrete event systems that are described by Petri nets. Our method is based on embedding the given Petri net model in a larger Petri net that retains the functionality and properties of the given one, perhaps in a *non-separate* (that is, not immediately identifiable) way. This *redundant Petri net embedding* introduces "structured redundancy" that can be used to facilitate fault detection, identification and correction, or to offer increased capabilities for monitoring and control. We focus primarily on *separate embeddings* in which the functionality of the original Petri net is retained in its exact form. Using these embeddings, we construct monitors that operate concurrently with the original system and allow us to detect and identify different types of failures by performing consistency checks between the state of the original Petri net and that of the monitor. The methods that we propose are attractive because the resulting monitors are robust to failures, they may not require explicit acknowledgments from each activity, and their construction is systematic and easily adaptable to restrictions in the available information. We also discuss briefly how to construct *non-separate* Petri net embeddings.

1 Introduction

In this paper we present a systematic methodology for providing monitoring capabilities to a given Petri net. Our approach consists of embedding the original Petri net in a *redundant* one (with more places, tokens and/or transitions) in a way that preserves the state, evolution and properties of the original Petri net in some encoded form[1]. We develop systematic ways of constructing monitoring schemes by focusing on the class of *separate Petri net embeddings*; these embeddings retain the functionality of the original system, but use additional places and tokens in order to impose invariant conditions that serve as consistency checks. The monitors operate concurrently with the original system and take actions based on the activity in the original system. By performing *linear* checks on the combined marking (state) of the original system and the monitor, the detecting mechanism is able to locate and identify failures in the overall system.

[1] These ideas are extensions of the algebraic techniques that were developed in [1,2] for protecting group/semigroup operations.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 188–207, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Our constructions automatically point out the additional connections that are necessary in order to allow fault monitoring. Furthermore, they can be adapted to incorporate changes in the configuration of the original system or to impose restrictions in the information that is available to the monitors (e.g. in cases where information about activity in parts of the system may be unavailable or corrupted).

The paper is organized as follows. In Section 2 we discuss the different types of failures that we will be protecting against. In Section 3 we construct schemes for failure detection and identification in Petri nets using separate redundant embeddings and in Section 4 we discuss more general Petri net embeddings. In Section 5 we describe how our embeddings can be used to facilitate control or detect illegal behavior in discrete event systems.

2 Fault Models

Let $\mathcal{S}$ be a Petri net with n places (p_1, p_2, ..., p_n) and m transitions (t_1, t_2, ..., t_m). Let b_{ij}^- denote the integer weight of the arc from place p_i to transition t_j, and b_{ij}^+ denote the integer weight of the arc from transition t_j to place p_i and define $\mathbf{B}^- = [b_{ij}^-]$ (respectively, $\mathbf{B}^+ = [b_{ij}^+]$) to be the $n \times m$ matrix with b_{ij}^- (respectively, b_{ij}^+) at its ith row, jth column position. The state evolution of Petri net $\mathcal{S}$ is given by

$$\mathbf{q}[k+1] = \mathbf{q}[k] + (\mathbf{B}^+ - \mathbf{B}^-)\mathbf{x}[k] \tag{1}$$

$$= \mathbf{q}[k] + \mathbf{B}\mathbf{x}[k]\ , \tag{2}$$

where $\mathbf{B} \equiv \mathbf{B}^+ - \mathbf{B}^-$ and $\mathbf{q}[k]$ is the state (*marking*) of the Petri net at time epoch k (i.e. the number of tokens in each of its places), [3]. The input $\mathbf{x}[k]$ in the above description is restricted to have exactly one non-zero entry with value 1. When $\mathbf{x}[k] = \mathbf{x}_j = \begin{bmatrix} 0 \cdots 1 \cdots 0 \end{bmatrix}^T$ (the 1 being at the jth position), transition t_j fires (j is in $\{1, 2, ..., m\}$). Note that transition t_j is enabled at time epoch k if and only if $\mathbf{q}[k] \geq \mathbf{B}^-(:, j)$ (where $\mathbf{B}^-(:, j)$ denotes the jth column of $\mathbf{B}^-$ and the inequality is taken element-wise). A *pure* Petri net is one in which no place serves as both an input and an output for the same transition (i.e. only one of b_{ij}^+ and b_{ij}^- can be non-zero). The Petri net in Figure 1 (with the indicated $\mathbf{B}^+$ and $\mathbf{B}^-$ matrices) is a pure Petri net.

Petri nets are a graphical and mathematical model for a variety of discrete event systems (DES's) including information and processing systems, [3]. Due to their power and flexibility, Petri nets are particularly relevant to the study of concurrent, asynchronous, distributed, nondeterministic, and/or stochastic systems, [4,5,6]. The extended spectrum of applications, their size and distributed nature, and the diverse implementations involved in modern Petri nets necessitate elaborate control, fault detection and recovery mechanisms.

We discuss three different error models which allow us to abstract away from the particulars of an implementation and the failures associated with it. Based on these error models, we develop a very general approach to fault tolerance. The

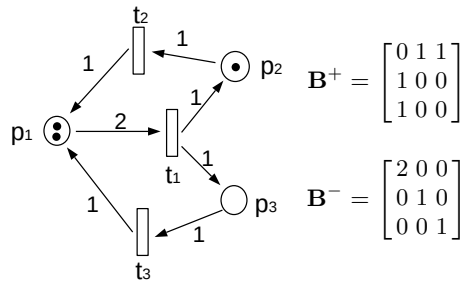


Fig. 1. A Petri net with three places and three transitions.

price that we pay for this generality is the fact that, given a particular system and the corresponding Petri net model, we need to ensure that our error model effectively captures the expected failures.

- A *transition failure* models a fault in the implementation of a certain transition. We say that transition t_j has a *postcondition failure* if no tokens are deposited to its output places (even though tokens from the input places have been used). Similarly, we say that transition t_j has a *precondition failure* if the tokens that are supposed to be removed from the input places of the faulty transition are not removed (even though tokens are deposited at the corresponding output places). In terms of the state evolution in eq. (1), a failure at transition t_j corresponds to transition t_j firing, but its preconditions (given by the jth column of $\mathbf{B}^-$, $\mathbf{B}^-(:, j)$) or its postconditions (given by $\mathbf{B}^+(:, j)$) not taking effect[2].
- A *place failure* models faults that corrupt the number of tokens in a *single* place of the Petri net. In terms of eq. (1), a place failure at time epoch k causes the value of a single variable in the state vector $\mathbf{q}[k]$ to be incorrect. This error model is suitable for Petri nets that represent computational systems or finite state machines (e.g. single-bit errors corrupt a single place in the Petri net). It has appeared in earlier work that dealt with fault detection in pure Petri nets, [8,9].
- The *additive failure model* is based on explicitly enumerating all faults that we would like to be able to detect or protect against. The error is then modeled by its *additive* effect on the state vector $\mathbf{q}[k]$ of the Petri net. In particular, if fault $f(i)$ takes place at time epoch k, then $\mathbf{q}_f[k] = \mathbf{q}[k] + \mathbf{e}_{f(i)}$ where $\mathbf{q}_f[k]$ is the faulty state of the Petri net and $\mathbf{e}_{f(i)}$ is the *additive* effect of fault $f(i)$. If we can find *a priori* the additive effect $\mathbf{e}_{f(i)}$ for each fault $f(i)$ that we would like to protect against, then we can define an $n \times l$ error matrix $E = \left[\mathbf{e}_{f(1)} \middle| \mathbf{e}_{f(2)} \middle| \cdots \middle| \mathbf{e}_{f(l)} \right]$, where l is the total number of different

[2] A fault in which *both* the preconditions and the postconditions are not executed is indistinguishable from the transition not taking place at all. The motivation for the transition failure error model came from [7], although the faults mentioned there are captured best by *place failures*.

failures expected. Based on the matrix E, we can construct embeddings to protect our Petri net.

The additive error model captures both transition and place failures:

- The additive effect of a precondition failure in transition t_j is captured by the error vector $\mathbf{e}^-_{t(j)} = \mathbf{B}^-(:, j)$. Similarly, the additive effect of a postcondition failure in t_j is captured by the additive error vector $\mathbf{e}^+_{t(j)} = -\mathbf{B}^+(:, j)$.
- The corruption of the number of tokens in place p_i is captured by the additive error vector $\mathbf{e}_{p(i)} = c \times \left[0 \cdots 1 \cdots 0\right]^T$, where c is an integer that denotes the number of tokens that have been added and the only non-zero entry of vector $\left[0 \cdots 1 \cdots 0\right]^T$ appears at the ith position.

The additive error model can also capture the effects of multiple *independent* additive failures (i.e. failures whose additive effects do not depend on whether the other failures have taken place or not). For example, a precondition failure at transition t_j and an *independent* failure at place p_i will result in the additive error vector $(\mathbf{e}^-_{t(j)} + \mathbf{e}_{p(i)})$.

Note that both the additive and place failure error models are capable of modeling any failure that results in the corruption of the number of tokens in certain places of a given Petri net. For example, a failure that corrupts the number of tokens in three places can be seen as three simultaneous place errors or as one additive error (associated *explicitly* with the effects of this failure). As in all modeling problems, the goal when choosing a particular error model should be to capture the effects of the underlying physical causes while at the same time allowing easy algebraic manipulation.

Example 1: Consider the Petri net in Figure 2. It could be the model of a distributed processing network or of a flexible manufacturing system. Transition t_2 models a process that takes as input two data packets (or two raw products) from place p_2 and produces two different data packets (or intermediate products), one of each being deposited to places p_3 and p_4. Processes t_3 and t_4 take their corresponding input packets (from places p_3 and p_4, respectively) to produce the final data packets (or final products). Note that processes t_3 and t_4 can take effect concurrently. Once done, they return separate acknowledgments to places p_5 and p_6 so that process t_2 can be enabled again. Transition t_1 models the external input to the system and is always enabled. The marking of the Petri net shown in Figure 2 is given by $\mathbf{q}[0] = \left[2\ 2\ 0\ 0\ 1\ 1\right]^T$; only transitions t_1 and t_2 are enabled.

If the process modeled by transition t_2 fails to execute its postconditions, tokens will be removed from input places p_2, p_5 and p_6, but no tokens will be deposited at output places p_3 and p_4. The faulty state of the Petri net will be $\mathbf{q}_f[1] = \left[2\ 0\ 0\ 0\ 0\ 0\right]^T$.

If process t_2 fails to execute its preconditions, then tokens will appear at the output places p_3 and p_4 but no tokens will be removed from the input places p_2, p_5 and p_6. The faulty state of the Petri net will be $\mathbf{q}_f[1] = \left[2\ 2\ 1\ 1\ 1\ 1\right]^T$.

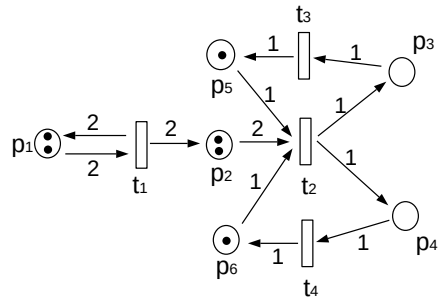


Fig. 2. A Petri net model of a distributed processing system.

If process t_2 executes correctly but there is a failure at place p_4, then the resulting state will be of the form $\mathbf{q}_f[1] = \begin{bmatrix} 2 & 0 & 1 & 1+c & 0 & 0 \end{bmatrix}^T$; the number of tokens at place p_4 has been corrupted by c. □

3 Monitoring Schemes

3.1 Separate Redundant Embeddings

We begin our study of *redundant embeddings* for Petri nets by considering *separate embeddings*[3]. The resulting monitors operate concurrently with the original system and are driven by the same input, i.e. are based on information about which transitions fire in the original system. By performing a check on the combined marking of the original system and the monitor, the detecting mechanism is able to locate and identify failures in the overall system. We develop systematic constructions that are based on *linear* checks (which makes them easily adaptable to changes in the configuration or the initial marking of the original Petri net). Furthermore, our schemes can be applied to systems where certain information is unavailable (e.g. when *no* connections are available from a particular place or transition). The alternative to our construction could be an analysis that is based on identifying invalid states (i.e. states reached only when a transition or a place fails) and then identifies the failure that has caused the Petri net to reach this invalid state. Our approach avoids this complicated reachability analysis, automatically points out additional connections that are necessary, and results in monitors that are robust to communication failures.

Definition 1: A *separate redundant embedding* for Petri net $\mathcal{S}$ (with n places, m transitions, state vector $\mathbf{q}[\cdot]$ and state evolution as in eq. (1)) is a Petri net

[3] In Section 4 we study a more general class of Petri net embeddings (which actually includes separate redundant embeddings as a special case). The reason we choose to present separate embeddings first is because they are easier to describe and because, unlike non-separate schemes, they can be applied to systems whose original structure (and the associated Petri net model) cannot be changed. Non-separate embeddings on the other hand may require changes in the structure of the original systems.

$\mathcal{H}$ (with $\eta \equiv n + d$, $d > 0$ places and m transitions) that has state evolution

$$\begin{aligned}\mathbf{q}_h[k+1] &= \mathbf{q}_h[k] + \mathcal{B}^+\mathbf{x}[k] - \mathcal{B}^-\mathbf{x}[k] \\ &= \mathbf{q}_h[k] + \begin{bmatrix}\mathbf{B}^+ \\ \mathbf{X}^+\end{bmatrix}\mathbf{x}[k] - \begin{bmatrix}\mathbf{B}^- \\ \mathbf{X}^-\end{bmatrix}\mathbf{x}[k]\,, \end{aligned} \tag{3}$$

and whose state vector is given by[4]

$$\mathbf{q}_h[k] = \underbrace{\begin{bmatrix}\mathbf{I}_n \\ \mathbf{C}\end{bmatrix}}_{\mathbf{G}}\mathbf{q}[k]$$

for all time epochs k. In addition, we require that for *any* initial marking (state) $\mathbf{q}[0]$ for $\mathcal{S}$, Petri net $\mathcal{H}$ (with initial state $\mathbf{q}_h[0] = \mathbf{G}\mathbf{q}[0]$) admits all firing transition sequences that are allowed in $\mathcal{S}$ (under initial state $\mathbf{q}[0]$). □

Note that the functionality of Petri net $\mathcal{S}$ remains intact within the redundant Petri net embedding $\mathcal{H}$. Matrix $\mathbf{G}$ is referred to as the *encoding* matrix. All valid states $\mathbf{q}_h[k]$ in $\mathcal{H}$ have to lie within the column space of $\mathbf{G}$; furthermore, there exists a parity check matrix $\mathbf{P} = \begin{bmatrix}-\mathbf{C} & \mathbf{I}_d\end{bmatrix}$ such that $\mathbf{P}\mathbf{q}_h[k] = \mathbf{0}$ for all k (under fault-free conditions). Since $\mathcal{H}$ is a Petri net, matrices $\mathbf{X}^+$ and $\mathbf{X}^-$ and state vector $\mathbf{q}_h[k]$ (for all k) have nonnegative integer entries. The following theorem characterizes separate redundant Petri net embeddings and leads to systematic ways of constructing them:

Theorem 1: Consider the setting described above. Petri net $\mathcal{H}$ is a separate redundant embedding for Petri net $\mathcal{S}$ (with state evolution as in eq. (1)) if and only if $\mathbf{C}$ is a matrix with nonnegative integer entries and

$$\mathbf{X}^+ = \mathbf{C}\mathbf{B}^+ - \mathbf{D}\,, \qquad \mathbf{X}^- = \mathbf{C}\mathbf{B}^- - \mathbf{D}\,,$$

where $\mathbf{D}$ is any $d \times n$ matrix with nonnegative integer entries such that $\mathbf{D} \leq \text{MIN}(\mathbf{C}\mathbf{B}^+, \mathbf{C}\mathbf{B}^-)$ (operations $\leq$ and MIN are taken element-wise). □

Proof: ($\Rightarrow$) The state $\mathbf{q}_h[0] = \mathbf{G}\mathbf{q}[0] = \begin{bmatrix}\mathbf{I}_n \\ \mathbf{C}\end{bmatrix}\mathbf{q}[0]$ needs to have nonnegative integer entries for all valid $\mathbf{q}[0]$ (a valid initial state for $\mathcal{S}$ is any vector $\mathbf{q}[0]$ with nonnegative integer entries). Clearly, a necessary (and sufficient) condition is that $\mathbf{C}$ is a matrix with nonnegative integer entries.

If we combine the state evolution of the redundant and original Petri nets (in eqs. (3) and (1), respectively) we see that

$$\begin{aligned}\mathbf{G}\mathbf{q}[k+1] \equiv \mathbf{q}_h[k+1] &= \mathbf{q}_h[k] + \begin{bmatrix}\mathbf{B}^+ \\ \mathbf{X}^+\end{bmatrix}\mathbf{x}[k] - \begin{bmatrix}\mathbf{B}^- \\ \mathbf{X}^-\end{bmatrix}\mathbf{x}[k] \\ &\Rightarrow \\ \begin{bmatrix}\mathbf{I}_n \\ \mathbf{C}\end{bmatrix}\mathbf{q}[k+1] &= \begin{bmatrix}\mathbf{I}_n \\ \mathbf{C}\end{bmatrix}\mathbf{q}[k] + \begin{bmatrix}\mathbf{B}^+ \\ \mathbf{X}^+\end{bmatrix}\mathbf{x}[k] - \begin{bmatrix}\mathbf{B}^- \\ \mathbf{X}^-\end{bmatrix}\mathbf{x}[k]\,.\end{aligned}$$

[4] More generally the state of a separate redundant embedding is given by $\mathbf{q}_h[k] = \begin{bmatrix}\mathbf{q}[k] \\ \phi(\mathbf{q}[k])\end{bmatrix}$ for an appropriate function ϕ.

Since any transition t_j can be enabled (e.g. by choosing $\mathbf{q}[0] \geq \mathbf{B}^-(:,j)$), we conclude that $\mathbf{X}^+ - \mathbf{X}^- = \mathbf{C}(\mathbf{B}^+ - \mathbf{B}^-)$. Without loss of generality we can set $\mathbf{X}^+ = \mathbf{C}\mathbf{B}^+ - \mathbf{D}$ and $\mathbf{X}^- = \mathbf{C}\mathbf{B}^- - \mathbf{D}$ for some matrix $\mathbf{D}$ with integer entries. In order for Petri net $\mathcal{H}$ (with initial marking $\mathbf{q}_h[0] = \left[\mathbf{q}^T[0]\ (\mathbf{C}\mathbf{q}[0])^T\right]^T$ where $\mathbf{q}[0]$ is *any* initial state for $\mathcal{S}$) to admit all firing transition sequences that are allowed in $\mathcal{S}$ under initial state $\mathbf{q}[0]$, we need $\mathbf{D}$ to have nonnegative integer entries. The proof follows easily by contradiction: suppose $\mathbf{D}$ has a negative entry in its jth column; choose $\mathbf{q}[0] = \mathbf{B}^-(:,j)$; transition t_j can be fired in $\mathcal{S}$ but cannot be fired in $\mathcal{H}$ because

$$\mathbf{C}\mathbf{q}[0] = \mathbf{C}\mathbf{B}^-(:,j) < \mathbf{C}\mathbf{B}^-(:,j) - \mathbf{D}(:,j) = \mathbf{X}^-(:,j)\ .$$

The requirement that $\mathbf{D} \leq \text{MIN}(\mathbf{C}\mathbf{B}^+, \mathbf{C}\mathbf{B}^-)$ follows from $\mathbf{X}^+$ and $\mathbf{X}^-$ being matrices with nonnegative integer entries.

($\Leftarrow$) The other direction follows easily. The only challenge is to show that if $\mathbf{D}$ is chosen to have nonnegative entries, then all transitions that are enabled in $\mathcal{S}$ at time epoch k under state $\mathbf{q}[k]$ are also enabled in $\mathcal{H}$ under state $\mathbf{q}_h[k] = \left[\mathbf{q}^T[k]\ (\mathbf{C}\mathbf{q}[k])^T\right]^T$: if $\mathbf{D}$ has nonnegative entries then

$$\begin{aligned}
\mathbf{q}[k] \geq \mathbf{B}^-(:,j) &\Rightarrow \mathbf{G}\mathbf{q}[k] \geq \mathbf{G}\mathbf{B}^-(:,j) \\
&\Rightarrow \mathbf{q}_h[k] \geq \mathbf{G}\mathbf{B}^-(:,j) \\
&\Rightarrow \mathbf{q}_h[k] \geq \mathbf{G}\mathbf{B}^-(:,j) - \begin{bmatrix} \mathbf{0} \\ \mathbf{D}(:,j) \end{bmatrix} \\
&\Rightarrow \mathbf{q}_h[k] \geq \mathcal{B}^-(:,j)\ .
\end{aligned}$$

(Remember that matrices $\mathbf{G}$, $\mathbf{B}^+$, $\mathbf{B}^-$, and $\mathbf{D}$ have nonnegative integer entries.) We conclude that if transition t_j is enabled in $\mathcal{S}$ ($\mathbf{q}[k] \geq \mathbf{B}^-(:,j)$), then it is also enabled in $\mathcal{H}$ ($\mathbf{q}_h[k] \geq \mathcal{B}^-(:,j)$). □

3.2 Failure Detection and Identification

Given a Petri net $\mathcal{S}$ with state evolution equation as in eq. (1), we can use a separate redundant embedding $\mathcal{H}$ as described in Theorem 1 to construct monitors for both transition and place failures. The invariant conditions imposed by our separate embeddings can be checked by verifying that $\left[-\mathbf{C}\ \mathbf{I}_d\right] \mathbf{q}_h[k]$ is equal to $\mathbf{0}$. The d additional places in $\mathcal{H}$ function as *checkpoint places* and can either be distributed in the Petri net system or be part of a centralized monitor[5]. In what follows, we show that by appropriately choosing $\mathbf{C}$ and $\mathbf{D}$ (subject to the constraints described in Theorem 1) we can detect and locate transition and/or place failures.

Transition Failures: Suppose that at time epoch $k-1$ transition t_j fires (that is, $\mathbf{x}[k-1] = \mathbf{x}_j$). If, due to a failure, the *postconditions* of transition t_j are not

[5] Some of the constraints that we developed in the previous section can be dropped if we adopt the view in [9] and treat additional places only as *test places*, i.e. allow them to have a negative number of tokens. In such case, $\mathbf{C}$ and $\mathbf{D}$ are not restricted to have nonnegative entries.

executed, the erroneous state at time epoch k will be

$$\mathbf{q}_f[k] = \mathbf{q}_h[k] - \mathcal{B}^+(:, j) = \mathbf{q}_h[k] - \mathcal{B}^+\mathbf{x}_j$$

(where $\mathbf{q}_h[k]$ is the state that would have been reached under fault-free conditions). The error syndrome will be

$$\begin{aligned}\mathbf{P}\mathbf{q}_f[k] &= \mathbf{P}(\mathbf{q}_h[k] - \begin{bmatrix}\mathbf{B}^+ \\ \mathbf{C}\mathbf{B}^+ - \mathbf{D}\end{bmatrix}\mathbf{x}_j) \\ &= \mathbf{D}\mathbf{x}_j \equiv \mathbf{D}(:, j)\ .\end{aligned}$$

If the *preconditions* of transition t_j are not executed, the erroneous state will be

$$\mathbf{q}_f[k] = \mathbf{q}_h[k] + \mathcal{B}^-(:, j) = \mathbf{q}_h[k] + \mathcal{B}^-\mathbf{x}_j$$

and the error syndrome will be

$$\mathbf{P}\mathbf{q}_f[k] = -\mathbf{D}\mathbf{x}_j \equiv -\mathbf{D}(:, j)\ .$$

If we choose all columns of $\mathbf{D}$ to be distinct, we will be able to detect and identify all single transition failures. Depending on the sign, we can also decide whether preconditions or postconditions were not executed. In fact, given enough redundancy, we may be able to identify multiple transition failures.

Example 2: Consider the Petri net in Figure 1, with the indicated $\mathbf{B}^+$ and $\mathbf{B}^-$ matrices. We will use one additional place ($d = 1$) in order to concurrently detect and identify transition failures. We choose $\mathbf{C} = \begin{bmatrix}2\ 2\ 1\end{bmatrix}$, $\mathbf{D} = \begin{bmatrix}3\ 2\ 1\end{bmatrix}$ and obtain the separate embedding of Figure 3 (the additional connections are shown with dotted lines).

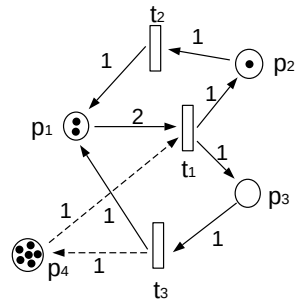


Fig. 3. A separate embedding to identify single transition failures in the Petri net of Figure 1.

Matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ are given by

$$\mathcal{B}^+ = \begin{bmatrix}\mathbf{B}^+ \\ \mathbf{C}\mathbf{B}^+ - \mathbf{D}\end{bmatrix} = \begin{bmatrix}0\ 1\ 1 \\ 1\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 0\ 1\end{bmatrix}\ , \mathcal{B}^- = \begin{bmatrix}\mathbf{B}^- \\ \mathbf{C}\mathbf{B}^- - \mathbf{D}\end{bmatrix} = \begin{bmatrix}2\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \\ 1\ 0\ 0\end{bmatrix}\ .$$

The parity check, performed concurrently by the monitoring mechanism, is given by

$$\left[-\mathbf{C}\ \mathbf{I}_1 \right] \mathbf{q}_h[k] = \left[-2\ -2\ -1\ 1 \right] \mathbf{q}_h[k]\ .$$

If the parity check is -3 (respectively -2, -1), then transition t_1 (respectively t_2, t_3) has failed to perform its preconditions. If the parity check is 3 (respectively 2, 1), then transition t_1 (respectively t_2, t_3) has failed to perform its postconditions.

The additional place p_4 is part of the monitoring mechanism: it receives information about the activity in the original Petri net (which transitions fire or complete, etc.) and appropriately updates the number of tokens in place p_4. The linear checker (not shown in Figure 3) concurrently detects and identifies failures by evaluating a checksum on the state of the overall system. Note that the monitoring mechanism in Figure 3 does not use any information about transition t_2. More generally, explicit connections from each transition to the monitoring mechanism may not be required; the scheme can also be adapted to handle cases where certain connections are not permitted (i.e. where information about a certain transition or place is not available). □

Place Failures: If, due to a failure, the number of tokens in place p_i is increased by c, the faulty state is given by

$$\mathbf{q}_f[k] = \mathbf{q}_h[k] + \mathbf{e}_{p(i)}$$

where $\mathbf{e}_{p(i)}$ is an η-dimensional vector with a unique non-zero entry at its ith position, i.e. $\mathbf{e}_{p(i)} = c \times \left[0 \cdots 1 \cdots 0 \right]^T$. In this case, the parity check will be

$$\begin{aligned} \mathbf{P}\mathbf{q}_f[k] &= \mathbf{P}(\mathbf{q}_h[k] + \mathbf{e}_{p(i)}) \\ &= c \times \mathbf{P}(:, i)\ . \end{aligned}$$

If we choose $\mathbf{C}$ so that columns of $\mathbf{P} \equiv \left[-\mathbf{C}\ \mathbf{I}_d \right]$ are not rational multiples of each other, then we can detect and identify single place failures[6].

Example 3: In order to concurrently detect and identify single place failures in the Petri net of Figure 1, we will use two additional places ($d = 2$) and choose $\mathbf{C} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \end{bmatrix}$. Note that the columns of the parity check matrix $\mathbf{P} = \left[-\mathbf{C}\ \mathbf{I}_2 \right]$ are not multiples of each other. Our choice for $\mathbf{D}$ is not critical in the identification of place failures and we choose[7] $\mathbf{D} = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}$. We then obtain a separate embedding

[6] We need to make sure that for all pairs of columns of $\mathbf{P}$ there do not exist non-zero integers α, β such that $\alpha \times \mathbf{P}(:, i) = \beta \times \mathbf{P}(:, j)$ $(i \neq j)$.

[7] This choice actually minimizes the number of additional connections (for the given choice of $\mathbf{C}$). See the discussion regarding how to choose matrices $\mathbf{C}$ and $\mathbf{D}$ at the end of this section.

with matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ given by

$$\mathcal{B}^+ = \begin{bmatrix} \mathbf{B}^+ \\ \mathbf{CB}^+ - \mathbf{D} \end{bmatrix} = \begin{bmatrix} 0\,1\,1 \\ 1\,0\,0 \\ 1\,0\,0 \\ 1\,0\,0 \\ 0\,1\,1 \end{bmatrix}, \mathcal{B}^- = \begin{bmatrix} \mathbf{B}^- \\ \mathbf{CB}^- - \mathbf{D} \end{bmatrix} = \begin{bmatrix} 2\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 2\,0\,0 \end{bmatrix}$$

and the parity check performed through

$$\begin{bmatrix} -\mathbf{C}\ \mathbf{I}_2 \end{bmatrix} \mathbf{q}_h[k] = \begin{bmatrix} -1\ -2\ -1\ 1\ 0 \\ -2\ -1\ -1\ 0\ 1 \end{bmatrix} \mathbf{q}_h[k]\,.$$

If the result is a multiple of $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ (respectively $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$), then place p_1 (respectively p_2, p_3, p_4, p_5) has failed. □

If $\mathbf{C}$ and $\mathbf{D}$ are chosen properly, we can actually perform detection and identification of both place and transition failures. Note that matrices $\mathbf{C}$ and $\mathbf{D}$ can be chosen almost independently (subject to the constraints analyzed above). The following example illustrates how this can be done.

Example 4: Concurrent identification of a single transition failure *or* a single place failure (but not both) in the Petri net of Figure 1 can be achieved with two additional places ($d = 2$). Let $\mathbf{C} = \begin{bmatrix} 3\,2\,3 \\ 2\,3\,3 \end{bmatrix}$ and $\mathbf{D} = \begin{bmatrix} 5\,2\,3 \\ 4\,1\,1 \end{bmatrix}$. With these choices, matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ are given by

$$\mathcal{B}^+ = \begin{bmatrix} \mathbf{B}^+ \\ \mathbf{CB}^+ - \mathbf{D} \end{bmatrix} = \begin{bmatrix} 0\,1\,1 \\ 1\,0\,0 \\ 1\,0\,0 \\ 0\,1\,0 \\ 2\,1\,1 \end{bmatrix}, \mathcal{B}^- = \begin{bmatrix} \mathbf{B}^- \\ \mathbf{CB}^- - \mathbf{D} \end{bmatrix} = \begin{bmatrix} 2\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,0\,0 \\ 0\,2\,2 \end{bmatrix}.$$

The parity check is performed through

$$\begin{bmatrix} -\mathbf{C}\ \mathbf{I}_2 \end{bmatrix} \mathbf{q}_h[k] = \begin{bmatrix} -3\ -2\ -3\ 1\ 0 \\ -2\ -3\ -3\ 0\ 1 \end{bmatrix} \mathbf{q}_h[k]\,.$$

If the parity check is a multiple of $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ (respectively $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$, $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$), then there is a failure in place p_1 (respectively p_2, p_3, p_4, p_5). If the parity check is $\begin{bmatrix} 5 \\ 4 \end{bmatrix}$ (respectively $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 3 \\ 1 \end{bmatrix}$), then transition t_1 (respectively t_2, t_3) has failed to perform its postconditions. If the parity check is $\begin{bmatrix} -5 \\ -4 \end{bmatrix}$ (respectively $\begin{bmatrix} -2 \\ -1 \end{bmatrix}$, $\begin{bmatrix} -3 \\ -1 \end{bmatrix}$), then transition t_1 (respectively t_2, t_3) has failed to perform its preconditions.

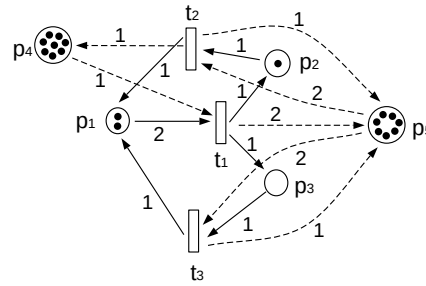


Fig. 4. A separate embedding to identify single transition or place failures in the Petri net of Figure 1.

The resulting Petri net embedding is shown in Figure 4 (the additional connections are shown with dotted lines; the linear checker is not shown in the figure). □

The graphical interpretation of the monitoring scheme in the above examples is straightforward: we add d places and connect them to the transitions of the original Petri net. The added places could be part of a centralized controller or could be distributed in the system. The tokens associated with the additional connections and places can be regarded as simple acknowledgment messages. The weights of the additional connections are given by the matrices $\mathbf{CB}^+ - \mathbf{D}$ and $\mathbf{CB}^- - \mathbf{D}$. The choice of matrix $\mathbf{C}$ specifies detection and identification for place failures, whereas the choice of $\mathbf{D}$ determines detection and identification for transition failures. Coding techniques or simple linear algebra can be used to guide the choice of $\mathbf{C}$ or $\mathbf{D}$. To detect single place failures we need to ensure that the columns of matrix $\mathbf{C}$ are not multiples of each other (this is what guided our choice of $\mathbf{C}$ in Examples 3 and 4). Similarly, matrices $\mathbf{D}$ in Examples 2 and 4 were chosen so that their columns are not the same (they *are* allowed to be multiples of each other).

The above discussion clearly demonstrates that, for given fault detection and identification requirements, there are many choices for matrices $\mathbf{C}$ and $\mathbf{D}$. One interesting future direction is to develop criteria for choosing among these different possibilities. Depending on the underlying system, plausible objectives could be to minimize the size of the monitor (number of additional places), the number of additional connections (from the original system to the additional places), or the number of tokens involved. Once these criteria are well-understood, it would be interesting to develop algorithmic techniques and automatic tools that allow us to systematically choose $\mathbf{C}$ and $\mathbf{D}$ so as to satisfy any or all of these criteria.

The additional places in our monitoring schemes (e.g. places p_4 and p_5 in Examples 2 and 4) may be parts of a centralized monitoring mechanism, so it may be reasonable in certain cases to assume that they are fault-free. Nevertheless, our scheme is capable of handling failures in any of the places in the Petri net embedding, including the added ones. The check mechanism (not shown in any

of the figures in Examples 2, 3 and 4) detects and identifies place failures by evaluating a checksum on the state of the overall Petri net embedding. Our implicit assumption has been that no failure take place during this checksum calculation. It would be interesting to investigate ways to relax this assumption.

Note that if we restrict ourselves to *pure* Petri nets, then we do not have a choice for $\mathbf{D}$. More specifically, we need to ensure that the resulting Petri net is pure which means that $\mathbf{D} = \text{MIN}(\mathbf{CB}^+, \mathbf{CB}^-)$. In such cases we may loose the ability to detect transition failures (we may attempt to treat them as *multiple* place failures). In this restricted case we recover the results in [8]: given a pure Petri net $\mathcal{S}$ as in eq. (2), we can construct a *pure* Petri net embedding with state evolution

$$\mathbf{q}_h[k+1] = \mathbf{q}_h[k] + \begin{bmatrix} \mathbf{B} \\ \mathbf{CB} \end{bmatrix} \mathbf{x}[k]$$

for a matrix $\mathbf{C}$ with nonnegative integer entries.

The distance measure adopted in [8] suggests that the redundant Petri net should guard against place failures (corruption of the number of tokens in individual places). The examples in [8] include a discussion of codes in finite fields and Petri nets in which addition is performed modulo some integer. (For example, modulo-2 addition matches well with Petri net systems in which places are implemented using binary memory elements. In this case, by choosing $\mathbf{P} \equiv \begin{bmatrix} -\mathbf{C} & \mathbf{I}_d \end{bmatrix}$ to be the (systematic) parity check matrix of a Hamming code [10], we can achieve single error detection and identification.)

4 Non-separate Redundant Embeddings

In this section we characterize more general redundant Petri net embeddings. This characterization can form the basis for systematically constructing more general monitoring schemes, but due to the lack of space we do not present the details or any examples; see [11] for more. Let $\mathcal{S}$ be a Petri net with n places, m transitions, and state evolution equation as given in eqs. (1) and (2); let $\mathbf{q}[0]$ be *any* initial state $\mathbf{q}[0] \geq \mathbf{0}$ and $\mathcal{X} = \{\mathbf{x}[0], \mathbf{x}[1], \ldots\}$ be *any* admissible (legal) firing sequence under this initial state.

Definition 2: Let $\mathcal{H}$ be a Petri net with $n+d$ places, $m+t$ transitions (where d, t are positive integers), initial state $\mathbf{q}_h[0]$, and state evolution equation

$$\begin{aligned} \mathbf{q}_h[k+1] &= \mathbf{q}_h[k] + \mathcal{B}^+\mathbf{z}[k] - \mathcal{B}^-\mathbf{z}[k] \\ &= \mathbf{q}_h[k] + (\mathcal{B}^+ - \mathcal{B}^-)\mathbf{z}[k] \,. \end{aligned} \tag{4}$$

Petri net $\mathcal{H}$ is a *redundant embedding* for $\mathcal{S}$ if it *concurrently simulates* $\mathcal{S}$ in the following sense: there exist

1. a *one-to-one* input encoding mapping $\xi : \mathbf{x}[k] \longmapsto \mathbf{z}[k]$,
2. a decoding mapping $\ell : \mathbf{q}_h[k] \longmapsto \mathbf{q}[k]$, and
3. an encoding mapping $g : \mathbf{q}[k] \longmapsto \mathbf{q}_h[k]$,

such that, for any initial state $\mathbf{q}[0]$ in $\mathcal{S}$ and any admissible firing sequence $\mathcal{X}$ (for $\mathbf{q}[0]$) and $\mathbf{z}[k] = \xi(\mathbf{x}[k])$: $\mathbf{q}[k] = \ell(\mathbf{q}_h[k])$ and $\mathbf{q}_h[k] = g(\mathbf{q}[k])$ for all time epochs $k \geq 0$. □

As defined above, a redundant embedding $\mathcal{H}$ is a Petri net that, after proper initialization (i.e. $\mathbf{q}_h[0] = g(\mathbf{q}[0])$), has the ability to *admit* any firing sequence $\mathcal{X}$ admissible by the original Petri net $\mathcal{S}$ (under *any* initial state $\mathbf{q}[0]$). The state of the original Petri net at any time epoch k is specified by the state of the redundant embedding (through mapping ℓ) and vice-versa (through mapping g). Note that, regardless of the initial state $\mathbf{q}[0]$ and the firing sequence $\mathcal{X}$, the state $\mathbf{q}_h[k]$ of the redundant embedding always lies in a subset of the redundant state space (namely, the image of $\mathbf{q}[\cdot]$ under the mapping g). The t additional transitions can be used for failure recovery.

For the rest of this section we focus on a special class of non-separate embeddings, where encoding and decoding can be performed through appropriate encoding and decoding *matrices*. Specifically, we consider the case where there exist an $n \times (n+d)$ decoding matrix $\mathbf{L}$ and an $(n+d) \times n$ encoding matrix $\mathbf{G}$ such that, under *any* admissible sequence of inputs $\mathcal{X} = \{\mathbf{x}[0], \mathbf{x}[1], \ldots\}$ and any initial state $\mathbf{q}[0]$: $\mathbf{q}[k] = \mathbf{L}\mathbf{q}_h[k]$ and $\mathbf{q}_h[k] = \mathbf{G}\mathbf{q}[k]$ for all time epochs $k \geq 0$. Furthermore, we will assume that $t = 0$ and (without loss of generality) treat ξ as the identity mapping (we can always permute the columns of $\mathbf{B}^+$ and $\mathbf{B}^-$ in eq. (1)). The state evolution equation of a non-separate redundant Petri net embedding is then given by

$$\mathbf{q}_h[k+1] = \mathbf{q}[k] + \mathcal{B}^+\mathbf{x}[k] - \mathcal{B}^-\mathbf{x}[k] \tag{5}$$

$$= \mathbf{q}_h[k] + \mathcal{B}\mathbf{x}[k] \ , \tag{6}$$

where $\mathcal{B} \equiv (\mathcal{B}^+ - \mathcal{B}^-)$.

The additional structure that is enforced through a redundant Petri net embedding of the form described above can be used for error detection and identification. In order to systematically construct and use redundant embeddings, we need to provide a common starting point. The following theorem characterizes Petri net embeddings in terms of a similarity transformation and a standard redundant Petri net:

Theorem 2: A Petri net $\mathcal{H}$ with $n+d$ places, m transitions and state evolution as in eqs. (5) and (6) is a redundant embedding for $\mathcal{S}$ (in eqs. (1) and (2)) only if it is *similar* (in the usual sense of change of basis in the state space, see [12]) to a *standard redundant embedding* $\mathcal{H}_\sigma$ whose state evolution equation is given by

$$\mathbf{q}_\sigma[k+1] = \mathbf{q}_\sigma[k] + \begin{bmatrix} \mathbf{B}^+ - \mathbf{B}^- \\ \mathbf{0} \end{bmatrix} \mathbf{x}[k] = \mathbf{q}_\sigma[k] + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{x}[k] \ . \tag{7}$$

Here, $\mathbf{B}^+$, $\mathbf{B}^-$ and $\mathbf{B} \equiv \mathbf{B}^+ - \mathbf{B}^-$ are the matrices in eqs. (1) and (2). Associated with the standard redundant embedding is the standard decoding matrix $\mathbf{L}_\sigma = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \end{bmatrix}$ and the standard encoding matrix $\mathbf{G}_\sigma = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix}$ (where $\mathbf{I}_n$ denotes the

$n \times n$ identity matrix). Note that the *standard* Petri net embedding is a *pure* Petri net. □

Proof: Clearly, $\mathbf{LGq}[\cdot] = \mathbf{Lq}_h[\cdot] = \mathbf{q}[\cdot]$. Since all initial states are possible in $\mathcal{S}$ ($\mathbf{q}[0]$ can be *any* vector with non-negative integer entries), we conclude that $\mathbf{LG} = \mathbf{I}_n$. In particular, $\mathbf{L}$ is full-row rank, $\mathbf{G}$ is full-column rank and there exists an $(n+d) \times (n+d)$ matrix $\mathcal{T}$ such that $\mathbf{L}\mathcal{T} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \end{bmatrix}$ and $\mathcal{T}^{-1}\mathbf{G} = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix}$, [12]. By employing the similarity transformation $\mathbf{q}'_h[k] = \mathcal{T}\mathbf{q}_h[k]$, we obtain a *similar* system $\mathcal{H}'$ whose state evolution is given by

$$\begin{aligned} \mathbf{q}'_h[k+1] &= (\mathcal{T}^{-1}\mathbf{I}_{(n+d)}\mathcal{T})\mathbf{q}'_h[k] + (\mathcal{T}^{-1}\mathcal{B})\mathbf{x}[k] \\ &\equiv \mathbf{q}'_h[k] + \mathcal{B}'\mathbf{x}[k] \,, \end{aligned}$$

and has decoding and encoding matrices $\mathbf{L}' = \mathbf{L}\mathcal{T} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \end{bmatrix}$ and $\mathbf{G}' = \mathcal{T}^{-1}\mathbf{G} = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix}$. Note that so far, we have not shown that system $\mathcal{H}'$ is necessarily a Petri net because the entries of $\mathcal{T}^{-1}\mathcal{B}$ are not guaranteed to be integers.

For all time epochs k, $\mathbf{q}'_h[k] = \mathbf{G}'\mathbf{q}[k] = \begin{bmatrix} \mathbf{q}[k] \\ \mathbf{0} \end{bmatrix}$; by combining the state evolution equations of the original Petri net and the redundant dynamic system we see that

$$\mathbf{q}'_h[k+1] = \mathbf{q}'_h[k] + \mathcal{B}'\mathbf{x}[k] \quad \Rightarrow \quad \begin{bmatrix} \mathbf{q}[k] + \mathbf{B}\mathbf{x}[k] \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{q}[k] \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathcal{B}'_1 \\ \mathcal{B}'_2 \end{bmatrix} \mathbf{x}[k] \,.$$

The above equations hold for all initial conditions $\mathbf{q}[0]$; since all transitions are enabled under some initial condition $\mathbf{q}[0]$, we see that $\mathcal{B}'_1 = \mathbf{B}$ and $\mathcal{B}'_2 = \mathbf{0}$.

If we regard the dynamic system $\mathcal{H}'$ as a pure Petri net, we see that any transition enabled in $\mathcal{S}$ is also enabled in $\mathcal{H}'$. Therefore, $\mathcal{H}'$ is a redundant Petri net embedding. In fact, it is the standard Petri net embedding $\mathcal{H}'$ with the decoding and encoding matrices presented in the theorem. □

The theorem provides a characterization of the class of Petri net embeddings for a given Petri net $\mathcal{S}$ and is a convenient starting point for systematically constructing redundant embeddings. For the standard Petri net $\mathcal{H}_\sigma$, the redundant invariant conditions that are imposed by the embedding are easily identified: since $\mathbf{q}_\sigma[\cdot] = \begin{bmatrix} \mathbf{q}[\cdot] & \mathbf{0} \end{bmatrix}^T$, these conditions are summarized by the *parity check* $\mathbf{P}_\sigma\mathbf{q}_\sigma[\cdot]$, where $\mathbf{P}_\sigma = \begin{bmatrix} \mathbf{0} & \mathbf{I}_d \end{bmatrix}$ is the parity check matrix. Note that the separate embeddings that were studied in Section 3 can also be put into the standard form of Theorem 2.

We now produce the converse to Theorem 2 which leads to the systematic construction of Petri net embeddings.

Theorem 3: Let $\mathcal{S}$ be a Petri net with n places, m transitions and state evolution as given in eqs. (1) and (2). A Petri net $\mathcal{H}$ with $n+d$ places, m transitions and state evolution as in eqs. (5) and (6) is a redundant embedding of $\mathcal{S}$ if:

- It is *similar* to a *standard* embedding $\mathcal{H}_\sigma$ (with state evolution equation as in (7)) through an $(n+d) \times (n+d)$ invertible matrix $\mathcal{T}$, such that the first n

columns of $\mathcal{T}^{-1}$ have *non-negative integer* entries. (The encoding, decoding and parity check matrices of the Petri net embedding $\mathcal{H}$ are then given by $\mathbf{L} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \end{bmatrix} \mathcal{T}$, $\mathbf{G} = \mathcal{T}^{-1} \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix}$, and $\mathbf{P} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_d \end{bmatrix} \mathcal{T}$.)

– Matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ are given by $\mathcal{B}^+ = \mathbf{G}\mathbf{B}^+ - \mathcal{D}$ and $\mathcal{B}^- = \mathbf{G}\mathbf{B}^- - \mathcal{D}$, where $\mathcal{D}$ is an $(n+d) \times m$ matrix with *non-negative integer* entries. Note that $\mathcal{D}$ has to be chosen so that the entries of $\mathcal{B}^+$ and $\mathcal{B}^-$ are *non-negative*, i.e. $\mathcal{D} \leq \text{MIN}(\mathbf{G}\mathbf{B}^+, \mathbf{G}\mathbf{B}^-)$. □

Proof: We know from Theorem 2 that any Petri net embedding $\mathcal{H}$ as in eqs. (5) and (6) can be obtained through an *appropriate* similarity transformation $\mathbf{q}_h[k] = \mathcal{T}\mathbf{q}_\sigma[k]$ of the standard embedding $\mathcal{H}_\sigma$ in eq. (7). In the process of constructing $\mathcal{H}$ from $\mathcal{H}_\sigma$, we need to ensure that $\mathcal{H}$ is a valid embedding for $\mathcal{S}$, i.e. we need to meet the following requirements:

1. Given any initial condition $\mathbf{q}[0]$ ($\mathbf{q}[0]$ has nonnegative integer entries), the state vector $\mathbf{q}_h[0] = \mathbf{G}\mathbf{q}[0] = \mathcal{T}^{-1} \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix} \mathbf{q}[0]$ should have non-negative integer entries.
2. Matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ should have *non-negative* integer entries.
3. The set of transitions enabled in $\mathcal{S}$ at any time epoch k should be a subset of the set of transitions enabled in $\mathcal{H}$ (so that under any initial condition $\mathbf{q}[0]$, a firing sequence $\mathcal{X}$ that is admissible in $\mathcal{S}$ is also admissible in $\mathcal{H}$).

The first condition has to be satisfied for any vector $\mathbf{q}[0]$ with non-negative integer entries. It is therefore necessary and sufficient that the first n columns of $\mathcal{T}^{-1}$ have non-negative integer entries. This also ensures that the matrix difference

$$\mathcal{B}^+ - \mathcal{B}^- = \mathcal{T}^{-1} \begin{bmatrix} \mathbf{B}^+ - \mathbf{B}^- \\ \mathbf{0} \end{bmatrix} = \mathcal{T}^{-1} \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix} \left(\mathbf{B}^+ - \mathbf{B}^-\right) \equiv \mathbf{G}(\mathbf{B}^+ - \mathbf{B}^-)$$

consists of integer entries. Without loss of generality we let $\mathcal{B}^+ = \mathbf{G}\mathbf{B}^+ - \mathcal{D}$, $\mathcal{B}^- = \mathbf{G}\mathbf{B}^- - \mathcal{D}$, where the entries of $\mathcal{D}$ are integers chosen so that $\mathcal{B}^+$ and $\mathcal{B}^-$ have non-negative entries (i.e. it is necessary that $\mathcal{D} \leq \mathbf{G}\mathbf{B}^+$ *and* $\mathcal{D} \leq \mathbf{G}\mathbf{B}^-$, where operation $\leq$ is taken element-wise).

We now check the last condition: transition t_j is enabled in the original Petri net $\mathcal{S}$ at time epoch k if and only if $\mathbf{q}[k] \geq \mathbf{B}^-(:, j)$. If $\mathcal{D}$ has non-negative entries then

$$\begin{aligned} \mathbf{q}[k] \geq \mathbf{B}^-\mathbf{x}_j &\Rightarrow \mathbf{G}\mathbf{q}[k] \geq \mathbf{G}\mathbf{B}^-\mathbf{x}_j \\ &\Rightarrow \mathbf{q}_h[k] \geq \mathbf{G}\mathbf{B}^-\mathbf{x}_j \\ &\Rightarrow \mathbf{q}_h[k] \geq (\mathbf{G}\mathbf{B}^- - \mathcal{D})\mathbf{x}_j \\ &\Rightarrow \mathbf{q}_h[k] \geq \mathcal{B}^-\mathbf{x}_j \ , \end{aligned}$$

where $\mathbf{B}^-(:, j) \equiv \mathbf{B}^-\mathbf{x}_j$ (recall that $\mathbf{q}[k]$, $\mathbf{B}^-$, $\mathbf{G}$ and $\mathcal{D}$ have *non-negative* integer entries). Therefore, if transition t_j is enabled in the original Petri net $\mathcal{S}$, it is

also enabled in $\mathcal{H}$ (transition t_j is enabled in $\mathcal{H}$ if and only if $\mathbf{q}_h[k] \geq \mathcal{B}^-(:, j)$). It is not hard to see that it is also *necessary* for $\mathcal{D}$ to have non-negative integer entries (otherwise we can find a counterexample by appropriately choosing the initial condition $\mathbf{q}[0]$). □

Note that no other restrictions are placed on a Petri net embedding. For example, the entries of the decoding matrix $\mathbf{L}$ and the parity check matrix $\mathbf{P}$ can be negative and/or rational.

5 Applications in Control

A discrete event system (DES) is usually monitored through a separate control mechanism that takes appropriate actions based on observations about the state and activity in the system. Control strategies (such as enabling or disabling transitions and external inputs) are often based on the Petri net that models the DES of interest, [13,14]. In this section we use the techniques of Section 3 to construct Petri net embeddings that allow the control mechanism to concurrently detect and locate multiple *active*[8] transitions. One of the biggest advantages of our approach is that it can be combined with failure detection and identification and that it is able to perform monitoring despite incomplete or erroneous information.

Example 5: We revisit the Petri net of Figure 2 which models a distributed processing network. If we add one extra place ($d = 1$) and use matrices $\mathbf{C} = \begin{bmatrix} 1 & 1 & 3 & 2 & 3 & 1 \end{bmatrix}$ and $\mathbf{D} = \begin{bmatrix} 2 & 5 & 3 & 1 \end{bmatrix}$, we obtain a separate Petri net embedding with one additional place that acts as a place-holder for special tokens (*acknowledgment tokens*): it receives 2 (1) such tokens whenever transition t_1 (t_4) is completed (i.e., we assume that our monitor mechanism observes the *completion* of transitions t_1 and t_4); it provides 1 token in order to enable transition t_2 (i.e., our monitoring mechanism observes the *initiation* of transition t_2). Explicit acknowledgments about the start and completion of all transition are avoided. Furthermore, by adding enough extra places, we can make the above monitoring scheme robust to incomplete or erroneous information (as in the case when a certain place is not required to or fails to submit the correct number of tokens).

At any given time epoch k, the controller of the redundant Petri net can identify if a transition is under execution by observing the state $\mathbf{q}_h[k]$ of the system and by performing the parity check

$$\begin{bmatrix} -\mathbf{C} & \mathbf{I}_1 \end{bmatrix} \mathbf{q}_h[k] = \begin{bmatrix} -1 & -1 & -3 & -2 & -3 & -1 & 1 \end{bmatrix} \mathbf{q}_h[k] \, .$$

(This of course assumes that each place can report the number of tokens stored in it.) If the result is 2 (5, 3, 1) then we know that transition t_1 (t_2, t_3, t_4) is under execution. Note that in order to identify whether multiple transitions are under execution, we need to use more extra places ($d > 1$). □

[8] We define an active transition as a transition that has used all tokens at its input places but has not yet returned any tokens at its output places (i.e. a transition that has not completed its postconditions).

One can also use separate embeddings to detect and identify illegal transitions in Petri net models. Suppose, for example, that the system modeled by the Petri net is "observed" through two different mechanisms: (i) *place sensors* that provide information about the number of tokens in a place, and (ii) *transition sensors* that indicate when a particular transition fires. One can construct a separate embedding that detects discrepancies in the information provided by these two sets of sensors in order to pinpoint illegal behavior.

Let the state evolution equation of the DES of interest be

$$\mathbf{q}[k+1] = \mathbf{q}[k] + \left[\, \mathbf{B}^+ \middle| \mathbf{B}_u^+ \,\right] \mathbf{x}[k] - \left[\, \mathbf{B}^- \middle| \mathbf{B}_u^- \,\right] \mathbf{x}[k] \;,$$

where the columns of $\mathbf{B}_u^+$ and $\mathbf{B}_u^-$ model the postconditions and preconditions of illegal transitions. We will then construct a separate embedding for the *legal* part of the network. The overall system will then have a state evolution

$$\mathbf{q}_h[k+1] \equiv \left[\begin{array}{c} \mathbf{q}_{h1}[k+1] \\ \hline \mathbf{q}_{h2}[k+1] \end{array}\right] = \mathbf{q}_h[k] + \left[\begin{array}{c|c} \mathbf{B}^+ & \mathbf{B}_u^+ \\ \hline \mathbf{CB}^+ - \mathbf{D} & \mathbf{0} \end{array}\right] \mathbf{x}[k] - \left[\begin{array}{c|c} \mathbf{B}^- & \mathbf{B}_u^- \\ \hline \mathbf{CB}^- - \mathbf{D} & \mathbf{0} \end{array}\right] \mathbf{x}[k] \;.$$

Our goal will be to choose $\mathbf{C}$ and $\mathbf{D}$ appropriately so that we can detect illegal behavior. Information about the state on the upper part of the embedding (vector $\mathbf{q}_{h1}[k+1]$) will be provided to the controller by the place sensors. The effect of illegal transitions will be captured in this part of the embedding by the changes in the number of tokens in the affected places. The additional places (with state vector $\mathbf{q}_{h2}[k]$) are internal to the controller and act only as test places[9]. Once the number of tokens in these test places is initialized appropriately (i.e. $\mathbf{q}_{h2}[0] = \mathbf{C}\mathbf{q}_{h1}[0]$), the controller removes or adds tokens to these places based on which transitions take place. Therefore, the information about the bottom part of the system (with state evolution $\mathbf{q}_{h2}[k+1] = \mathbf{q}_{h2}[k] + \left[\, \mathbf{CB}^+ - \mathbf{D} \middle| \mathbf{0} \,\right] \mathbf{x}[k] - \left[\, \mathbf{CB}^- - \mathbf{D} \middle| \mathbf{0} \,\right] \mathbf{x}[k]$) is (indirectly) provided by the transition sensors.

When an illegal transition fires during time epoch k, the illegal state $\mathbf{q}_f[k]$ of the redundant embedding is given by

$$\mathbf{q}_f[k] = \mathbf{q}_h[k] + \left[\begin{array}{c} \mathbf{B}_u^+ \\ \hline \mathbf{0} \end{array}\right] \mathbf{x}_u[k] - \left[\begin{array}{c} \mathbf{B}_u^- \\ \hline \mathbf{0} \end{array}\right] \mathbf{x}_u[k] \;,$$

where $\mathbf{x}_u[k]$ denotes a vector with all zero entries, except an entry that is 1 and is associated with the illegal transition that fired. If we perform the parity check $\mathbf{P}\mathbf{q}_f[k]$ we get:

$$\mathbf{P}\mathbf{q}_f[k] = \left[\, -\mathbf{C} \;\; \mathbf{I}_d \,\right] \mathbf{q}_f[k] = \ldots = -\mathbf{C}\mathbf{B}_u\mathbf{x}_u[k] \;.$$

Therefore, we can identify which illegal transition has fired if all columns of $\mathbf{CB}_u$ are unique.

[9] Test places cannot inhibit transitions and can have a negative number of tokens. A connection from a test place p_i to transition t_j indicates that the number of tokens in p_i will decrease when t_j fires.

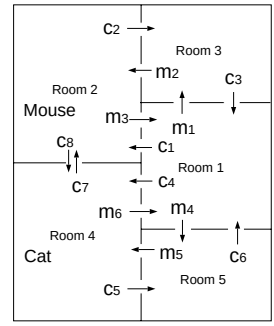


Fig. 5. The cat-and-mouse maze.

Example 6: Consider the Petri net version of the popular "cat-and-mouse" problem [13], originally introduced in the setting of supervisory control in [15]. We are given the maze of five rooms shown in Figure 5; a cat and a mouse circulate in this maze, with the cat moving from room to room through *unidirectional* "doors" $\{c_1, c_2, ..., c_8\}$ and the mouse through unidirectional "doors" $\{m_1, m_2, ..., m_6\}$. The Petri net model is based on two separate subnets, one dealing with the cat's position and movements and the other dealing with the mouse's position and movements. Each subnet has five places, corresponding to the five rooms in the maze. A token in a certain place indicates that the mouse (or the cat) is in the corresponding room. Transitions model the movement of the two animals between different rooms (as allowed by the structure of the maze in Figure 5). In particular, the subnet that deals with the mouse has a state vector with five variables, exactly one of which has the value 1 (the rest are set to 0). The state evolution for this subnet is given by eqs. (1) and (2) with

$$\mathbf{B}^+ = \begin{bmatrix} 0\,0\,1\,0\,0\,1 \\ 0\,1\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0 \end{bmatrix}, \mathbf{B}^- = \begin{bmatrix} 1\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,1\,0 \end{bmatrix}.$$

For example, $\mathbf{q}[k] = \begin{bmatrix} 0\,1\,0\,0\,0 \end{bmatrix}^T$ means that at time epoch k the mouse is in room 2. Transition t_3 takes place when the mouse moves from room 2 to room 1 through door m_3; this causes the new state to be $\mathbf{q}[k+1] = \begin{bmatrix} 1\,0\,0\,0\,0 \end{bmatrix}^T$.

We assume that the controller of the maze in Figure 5 obtains information about the state of the system through a set of detectors. More specifically, each room is equipped with a "mouse sensor" that indicates whether the mouse is in that room. In addition, door sensors get activated whenever the mouse goes through the corresponding door. Suppose that due to a bad choice of materials, the maze of Figure 5 is built in a way that allows the mouse to dig a tunnel connecting rooms 1 and 5 and/or a tunnel connecting rooms 1 and 4. The exis-

tence of such a tunnel leads to illegal (i.e. non-door) transitions in the network described by $\mathbf{B}_u = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 1 & 0 & 0 \end{bmatrix}$.

In order to detect the existence of such tunnels we can use a Petri net embedding with one additional place ($d = 1$), $\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 2 & 3 \end{bmatrix}$ and $\mathbf{D} = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 1 \end{bmatrix}$. The resulting redundant matrices $\mathcal{B}^+$ and $\mathcal{B}^-$ are given by

$$\mathcal{B}^+ = \left[\begin{array}{c|c} \mathbf{B}^+ & \mathbf{B}_u^+ \\ \hline \mathbf{CB}^+ - \mathbf{D} & \mathbf{0} \end{array}\right] = \left[\begin{array}{cccccc|cccc} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right], \mathcal{B}^- = \left[\begin{array}{cccccc|cccc} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}\right].$$

The upper part of the network is observed through the place sensors. The number of tokens in the additional place is updated based on information from the transition sensors (it receives 2 tokens when transition t_3 fires; it looses 1 token when t_4 or t_5 fires). The parity check is given by

$$\begin{bmatrix} -1 & -1 & -1 & -3 & -2 & -1 & 1 \end{bmatrix} \mathbf{q}_h[k]$$

and is 0 if no illegal activity has taken place. It is 2 (-2, 1, -1) if illegal transition $\mathbf{B}_u(:,1)$ ($\mathbf{B}_u(:,2)$, $\mathbf{B}_u(:,3)$, $\mathbf{B}_u(:,4)$) has taken place. □

6 Conclusion

In this paper we have investigated ways of systematically incorporating redundancy into a given Petri net in order to achieve failure monitoring or facilitate the control of the underlying DES. We defined and characterized particular classes of Petri net embeddings, which we then used for systematically constructing fault detection and identification schemes under different error models. Our approach extends the results on fault-tolerant Petri nets in [8,9] by introducing non-separate embeddings, by allowing the study of non-pure Petri nets, and by extending the methods to fault-tolerant monitoring schemes for DES. The resulting monitors use linear checks to detect and identify failures; furthermore, they are robust to erroneous or incomplete information and do not need to be re-constructed when the initial state of the Petri net changes. Our approach is very general and does not make any assumption about the underlying DES.

One interesting extension of our work would be towards the development of *robust control* schemes for DES's that are observed at a central or distributed controller through a network of remote and unreliable sensors. In such cases, we can use Petri net embeddings to design redundant sensor networks and to devise control strategies that achieve the desired objective despite the presence of uncertainty (in the sensors themselves or in the information received by the

controller). A related research direction would be to characterize different criteria for choosing our redundant embeddings (e.g. matrices **C** and **D**) in order to eventually develop tools that automatically provide fault detection and identification to DES's of interest (e.g. network protocols). Another possibility for future research is to develop more general redundant embeddings and to explicitly study examples where a subset of transitions is uncontrollable and/or unobservable (e.g. when links to or from the transition are not possible, [14]). Also, in order to achieve error correction, we need to introduce *error recovery transitions* which can serve as correction mechanisms when particular failures are detected. We are also considering ways to extend our approach to hierarchical or distributed monitoring schemes for large Petri nets by enforcing appropriate constraints on the corresponding embeddings.

References

1. P. E. Beckmann, Fault-Tolerant Computation Using Algebraic Homomorphisms. PhD thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1992.
2. C. N. Hadjicostis, "Fault-Tolerant Computation in Semigroups and Semirings," M. Eng. thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1995.
3. T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, April 1989.
4. F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Synchronization and Linearity*. New York: Wiley, 1992.
5. C. G. Cassandras, S. Lafortune, and G. J. Olsder, *Trends in Control: A European Prospective*. London: Springer-Verlag, 1995.
6. C. G. Cassandras, *Discrete Event Systems*. Boston: Aksen Associates, 1993.
7. V. Y. Fedorov and V. O. Chukanov, "Analysis of the fault tolerance of complex systems by extensions of Petri nets," *Automation and Remote Control*, vol. 53, no. 2, pp. 271–280, 1992.
8. J. Sifakis, "Realization of fault-tolerant systems by coding Petri nets," *Journal of Design Automation and Fault-Tolerant Computing*, vol. 3, pp. 93–107, April 1979.
9. M. Silva and S. Velilla, "Error detection and correction on Petri net models of discrete events control systems," in *Proceedings of the ISCAS*, pp. 921–924, 1985.
10. S. B. Wicker, *Error Control Systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.
11. C. N. Hadjicostis, Coding Approaches to Fault Tolerance in Dynamic Systems. PhD thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1999.
12. C. N. Hadjicostis and G. C. Verghese, "Structured redundancy for fault tolerance in state-space models and Petri nets," *Kybernetika*. To appear.
13. K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of Petri net based on place invariants," *Automatica*, vol. 32, no. 1, pp. 15–28, 1996.
14. J. O. Moody and P. J. Antsaklis, "Supervisory control using computationally efficient linear techniques: a tutorial introduction," in *Proceedings of 5th Mediterranean Conference on Control and Systems*, (Cyprus), July 1997.
15. P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, pp. 81–97, 1989.

Quasi-static Scheduling of Embedded Software Using Equal Conflict Nets

Marco Sgroi[1], Luciano Lavagno[2], Yosinori Watanabe[3], and Alberto Sangiovanni-Vincentelli[1]

[1] University of California at Berkeley, EECS Department, Berkeley, CA 94720
[2] Cadence Berkeley Labs, 2001 Addison Street, Berkeley, CA 94704-1144
[3] Cadence European Labs, via S.Pantaleo 66, 00186 Roma, Italy

Abstract. Embedded system design requires the use of efficient scheduling policies to execute on shared resources, e.g. the processor, algorithms that consist of a set of concurrent tasks with complex mutual dependencies. Scheduling techniques are called *static* when the schedule is computed at compile time, *dynamic* when some or all decisions are made at run-time. The choice of the scheduling policy mainly depends on the specification of the system to be designed. For specifications containing only data computation, it is possible to use a fully static scheduling technique, while for specifications containing data-dependent control structures, like the if-then-else or while-do constructs, the dynamic behaviour of the system cannot be completely predicted at compile time and some scheduling decisions are to be made at run-time. For such applications we propose a Quasi-static scheduling (QSS) algorithm that generates a schedule in which run-time decisions are made only for data-dependent control structures. We use Equal Conflict (EC) nets as underlying model, and define quasi-static schedulability for EC nets. We solve QSS by reducing it to a decomposition of the net into conflict-free components. The proposed algorithm is complete, in that it can solve QSS for any EC net that is quasi-statically schedulable.

1 Introduction

Embedded systems are informally defined as a collection of programmable units, e.g. microcontrollers and DSPs, surrounded by ASICs and other standard components, that interact with the environment through sensors and actuators. Software development and its integration with the hardware is one of the main sources of cost in embedded system design. Modern design flows allow the designer to start with a functional specification of the overall system and map it onto a heterogeneous architecture including processors, memories and ASICs. However, an implementation on a shared resource, such as a processor, requires one to solve a *scheduling* problem, in order to sequence the execution of the concurrent modules while simultaneously (1) satisfying real-time constraints and (2) using the processor and the memory resources as efficiently as possible.

Embedded systems specifications usually contain both data computations and control structures. Control structures can be of two types: (1) data-dependent

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 208–227, 1999.
© Springer-Verlag Berlin Heidelberg 1999

controls, like if-then-else or while-do loops, determine the next operation to be executed by testing the *value* of some data, and (2) real-time controls, like preemption and suspension, trigger actions after the *occurrence* of external or internal events. For specifications containing only data computations (assignments and fixed iteration loops) the schedule can be completely computed at compile time, and therefore is called *static*. A static schedule, usually implemented as a single task, is predictable and can be executed with almost no run-time overhead. However, when specifications include also some control it is not possible to compute the entire schedule at compile time. If the specification contains only data-dependent type of control in addition to data computation, the order in which operations are executed depends on the value of some data, which is known at run-time. In this case, *quasi-static* scheduling [1] techniques compute most of the schedule at compile time, leaving at run-time only the solution of data-dependent choices. Furthermore, if the specification allows communication via queues of unbounded size (e.g., in SDL or Dataflow networks), (quasi)static scheduling can bound the maximum size of those queues and ensure correct execution on an embedded system with a finite amount of physical memory. For specifications containing also real-time controls, the run-time behaviour heavily depends on the occurrence of external events. In this case, classical Real-Time scheduling techniques can be used to decide at run-time which tasks should be executed in reaction to such events. This type of schedule is called *dynamic*.

In this paper we present a new approach to schedule specifications including not only data-processing computations but also data-dependent control structures. We propose a *Quasi-static scheduling* (QSS) algorithm that takes as input a Petri Net (PN) model of the specification and produces as output a software implementation consisting of a set of concurrent tasks. The scheduling technique is quasi-static in the sense that, although any control decision is still made at run-time based on the value of the data, fragments of code that need to be executed as the consequence of the resolution are scheduled at compile time. Our technique addresses the problem of partitioning the functionality of the specification into tasks, i.e. functional blocks having the same execution rate. In particular, the algorithm we propose finds automatically a partition with minimum number of tasks, which is a problem usually solved by hand by experienced designers, and therefore allows to significantly reduce the run-time overhead and improve performances in single processor architectures. Here we address the problem of identifying tasks and deriving the code for each task, while issues like real-time dynamic scheduling of tasks are out of the scope of this paper.

We have chosen PNs as underlying formal model, because they allow to express concurrency, non-deterministic choice, synchronization and causality and because most properties are decidable for PNs. We represent data computations using *transitions* and channels between computation units using *places*. Data-

[1] Static and quasi-static scheduling generates sequential code and statically allocates communication buffers. Hence it is often called "software synthesis" in the literature, as opposed to the term "scheduling", that is often reserved to dynamic real-time scheduling.

dependent control is modeled by places, called *choices*, with multiple output transitions, one for each possible resolution of the control. Data are modeled as tokens passed by transitions through places. In particular, we use a sub-class of PNs called *Equal Conflict Nets* (EC nets), because they exhibit clear distinction between the notions of concurrency and choice. Hence, they are appropriate to model computations in which control decisions depend on the value rather than on the arrival time of a token [2].

In this paper we introduce the notion of schedulability for EC nets. Informally, an EC net is quasi-statically schedulable if for *every resolution* of the control at the choice places, there exists a finite *cyclic firing sequence* that returns the tokens of the net to their initial places. The existence of cyclic sequences is required because it ensures that the number of data tokens that accumulate in any place is bounded even for infinite execution. We present an algorithm that first checks schedulability of the net to verify the correctness of the specification. If the net is not schedulable, the designer is notified that there exists no implementation that can be executed forever with bounded memory. If the net is schedulable, the algorithm computes a quasi-static schedule by *decomposing* the net into conflict free components and then applies static scheduling techniques to each component.

Previous work has used the decomposition of Free-Choice (FC) PNs into conflict free components. The fundamental theorem of Hack [9] on live and safe strongly connected FCPNs is based on the decomposition of the net into as many Marked Graphs (MGs) reductions as the number of the possible allocations of the non-deterministic choices. In [3] Best proposes an iterative algorithm to decompose a strongly connected ordinary PN into a set of strongly connected MGs. More recently, Teruel in [6] extends to weighted nets known results for ordinary nets [1]. These works have their main application in checking whether a given strongly connected net is bounded. However, in the domain of embedded reactive systems applications usually have lots of interactions with the environment, that are naturally modeled as source and sink transitions. As a result, *nets modeling embedded systems are not strongly connected.* Moreover, boundedness is a too restrictive property for our objective, that is finding *one* implementation with guaranteed bounded memory execution. In fact, schedulability implies the *existence* of at least one valid schedule that ensures that there is no unbounded accumulation of tokens in any place, while boundedness implies that *for all* the reachable markings, the number of tokens in any place does not exceed a certain number k. Therefore, we modify Hack's MG decomposition algorithm and apply it to the class of EC nets that have source and sink transitions.

Finally, our technique derives a software implementation of the schedule by traversing it and replacing transitions with the corresponding code (inline coding). A different approach to software synthesis using PNs has been recently proposed [8]. The algorithm in [8] generates a software program from a concurrent process specification through an intermediate PN representation. This

[2] EC nets model data-dependent control by abstracting the if-then-else control decisions as non-deterministic choices.

approach is based on the strong assumption that the PN is safe, i.e. buffers can store at most one data unit. This on one side guarantees termination of the algorithm, on the other side it makes impossible to handle multirate specifications, like FFT computations and downsampling. Moreover, safeness excludes the possibility to use PNs where source and sink transitions model the interaction with the environment: this makes it impossible to specify inputs with independent rate [3]. This method, assuming a priori that the net is schedulable, does not address also the important problem of verifying the schedulability of the specification.

The paper is organized as follows. In Section 2 we recall some definitions of the PN model and in Section 3 we shortly describe known techniques for scheduling of Weighted T-Systems. Section 4 defines the Quasi-Static Scheduling problem for EC nets and presents an algorithm to find a solution, if there exists one. Then, we describe how to generate a C program in Section 5. Section 6 presents an ATM Server application and experimental results.

2 Petri Nets: Background

A **Petri Net** is a triple (P, T, F), where P is a non-empty finite set of places, T a non-empty finite set of transitions and $F : (T \times P) \cup (P \times T) \rightarrow \mathbb{N}$ the weighted flow relation between transitions and places. A **Petri Net graph** is a representation of a Petri Net as a bipartite weighted directed graph. If $F(x, y) > 0$, there is an arc with weight $F(x, y)$ from node x to node y. Given a node x, either a place or a transition, its **preset** is defined as ${}^\bullet x = \{y | (y, x) \in F\}$ and its **postset** as $x^\bullet = \{y | (x, y) \in F\}$. For a node y, $Pre[X, y]$ is a vector whose i-th component is equal to $F(x_i, y)$. A transition (place) whose preset is empty is called **source transition** (place), a transition (place) whose **postset** is empty is called **sink transition** (place). A place p such that $|p^\bullet| > 1$ is called **choice** or **conflict**. If $|{}^\bullet p| > 1$, p is called **merge**. Two transitions t and t' of a net N are said to be in **Equal Conflict Relation** [4] if $Pre[P, t] = Pre[P, t'] \neq 0$. A **marking** μ is an n-vector $\mu = (\mu_1, \mu_2, ..., \mu_n)$ where $n = |P|$ and μ_i is the non-negative number of tokens in place p_i. A transition t such that each input place p_i is marked with at least $F(p_i, t)$ tokens is enabled and may fire. When transition t fires, $F(p_i, t)$ tokens are removed from each input place p_i and $F(t, p_j)$ tokens are produced in each output place p_j.

The following PN properties, that are decidable for any Petri Net [11], are relevant in our discussion: **1) Reachability**. A marking μ' is reachable from a marking μ if there exists a firing sequence σ starting at marking μ and finishing at μ'. **2) Boundedness**. A Petri Net is said to be k-bounded if the number of tokens in every place of a rechable marking does not exceed a finite number k. A

[3] Two inputs have independent rate if their rates are not rationally related. An example of inputs with independent rate are the input keys from a keyboard. Instead, two streams of PCM samples for stereo audio are inputs with dependent rate.

[4] The Equal Conflict Relation is an equivalence relation that partitions the set of transitions of the net into a set of equivalence classes called Equal Conflict Sets [6].

safe Petri Net is one that is 1-bounded. **3) Deadlock-freedom**. A Petri Net is deadlock-free if, no matter what marking has been reached, it is possible to fire at least one transition of the net. **4) Liveness**. A Petri Net is live if for every reachable marking and every transition t it is possible to reach a marking that enables t.

Definition 1. *A PN is a* **Weighted (W)T-System** *if* $\forall p \in P : |p^\bullet| = |p^\bullet| = 1$.

Definition 2. *A PN is a* **Conflict Free (CF) Net** *if* $\forall p \in P : |p^\bullet| \leq 1$.

Definition 3. *A PN is an* **Equal Conflict (EC) Net** *if* ${}^\bullet t \cap t^\bullet \neq 0 \Rightarrow Pre[P,t] = Pre[P,t']$.

3 Static Scheduling of Weighted T-Systems

Weighted T-Systems (WT-Systems) [5] is a subclass of PNs that can represent concurrency and synchronization but not conflict and, therefore, is used to model specifications including pure data computation. WT-Systems can be mapped to Synchronous Dataflow networks where transitions are actors and places arcs; hence, one can apply to WT-Systems the well-known techniques for SDF scheduling [2]. The approach proposed by Lee [2] to find a static schedule for an SDF graph is based on the notion of finite complete cycle.

Definition 4. *Given a Petri Net and an initial marking, a* **finite complete cycle** *is a sequence of transition firings that returns the net to its initial marking.*

Since both the number of transition firings in a finite complete cycle and the tokens produced by each firing are finite, the number of tokens that can accumulate in any place of the net during the execution is bounded. If such a finite complete cycle exists, the net can be executed forever with bounded memory by repeating infinitely many times this sequence of transition firings (figure 1). Therefore, in [2] a static schedule is a periodic sequence of transitions and the period is a finite complete cycle.

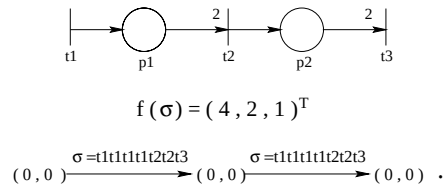


Fig. 1. Cyclic schedule

[5] Weighted T-Systems generalize Marked Graphs allowing arcs with non-unit weights

The problem of finding a finite complete cycle for a net can be reduced to the reachability problem, that is known to be decidable [11]. In terms of reachability the goal is to find a sequence of transitions σ that starting from the marking μ returns the net to the same marking μ. To find σ, one must first solve the state equations [11] $f(\sigma)^T \cdot D = 0$. Here, D is a matrix where the i-th row corresponds to a transition t_i, the j-th column corresponds to a place p_j, and $D[i,j] = F(t_i, p_j) - F(p_j, t_i)$. A solution $f(\sigma)$, called **T-invariant** [1], is a vector whose i-th component $f_i(\sigma)$ is the number of times that transition t_i appears in sequence σ.

Definition 5. *A Petri Net is* **consistent** *iff* $\exists f(\sigma) > 0$ *s.t.* $f(\sigma)^T \cdot D = 0$.

The existence of a T-invariant is a necessary, but not sufficient condition for a finite complete cycle to exist. In fact, even if there exists a solution of the state equations, deadlock can still occur if there are not enough tokens to fire any transition. Therefore, once a T-invariant $f(\sigma)$ is obtained, it is necessary to verify by simulation that there exists a sequence σ_i that contains transition t_j as many times as $f_j(\sigma)$ and such that the net does not deadlock during execution. Such a sequence σ_i, if it exists, is a finite complete cycle. Lee [2] has shown that it is sufficient to simulate the firing sequences corresponding to the minimal vector [6] in the one-dimensional T-invariant space. This approach can be adopted for WT-Systems, but it is not adequate for larger classes of PNs containing non-deterministic choices [7].

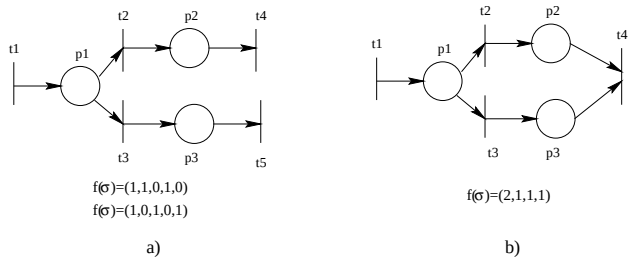


Fig. 2. Schedulable (a) and not schedulable (b) EC nets

[6] A T-invariant is **minimal** when its set of non-zero entries is not a strict superset of that of any other T-invariant and the greatest common divisor of its elements is one [6].

[7] Nets containing non-deterministic choices may have two or more T-invariants that are linearly-independent (figure 2a).

4 Quasi-static Scheduling of ECN

4.1 Definition of Schedulability

Let $\Sigma = \{\sigma_1, \sigma_2 ...\}$ be a non-empty finite set of finite firing sequences such that for all $\sigma_i \in \Sigma$, σ_i is a finite complete cycle. Let σ_i^j be the j-th transition in sequence $\sigma_i = (\sigma_i^1 \sigma_i^2 ... \sigma_i^{j-1} \sigma_i^j \sigma_i^{j+1} ... \sigma_i^{N_i})$ and let Θ be the characteristic function of the Equal Conflict Relation, i.e. $\Theta(t, t') = 1$ iff t and t' are in Equal Conflict Relation.

Definition 6. *The set Σ is a* **valid set of finite complete cycles** *if:*
(1) $\forall \sigma_i \in \Sigma$, $\forall \sigma_i^j \in \sigma_i$ s.t. $\sigma_i^j \neq \sigma_i^h$ $\forall h < j$, $\forall t_k \in T$ s.t. $t_k \neq \sigma_i^j$ and $\Theta(t_k, \sigma_i^j) = 1$, $\exists \sigma_l \in \Sigma$ s.t.
(a) $\sigma_l^m = \sigma_i^m$, $\forall m \leq j-1$
(b) $\sigma_l^m = t_k$, $m{=}j$
(2) $\forall \sigma_i \in \Sigma$, there exists no transition $t \notin \sigma_i$ that is always enabled when σ_i is executed (fairness).

This definition informally means that for each sequence σ_i that includes a conflict transition σ_i^j, for each transition t_k that is in Equal Conflict Relation with σ_i^j, there exists another sequence σ_l s.t. σ_i and σ_l are identical up to the (j-1)th transition and have respectively σ_i^j and t_k at the j-th position in the sequence. Condition 2 (fairness) guarantees that there is no transition that, although always enabled during a cycle, is never executed. This definition allows for some transitions to be dead; liveness checking is trivially done once T-invariants have been derived.

Definition 7. *Given an EC net N and an initial marking μ_0, a valid set of finite complete cycles Σ is* **executable** *if the net does not deadlock when its execution is simulated.*

Definition 8. *A valid set of finite complete cycles Σ is a* **valid schedule** *if it is executable.*

Definition 9. *Given an EC net N* [8] *and an initial marking μ_0, the pair (N,μ_0) is* **(quasi-statically) schedulable**, *if there exists a valid schedule.*

This definition of schedulability extends to EC nets the concept of WT-System scheduling given in Section 3. If the net contains non-deterministic choices that model data dependent structures like if-then-else or while-do, a valid schedule is an executable set of firing sequences, one for every resolution of non-deterministic choices. A valid schedule must contain a finite complete cycle for every possible outcome of a choice because the value of the control tokens is unknown at compile time when the valid schedule is computed.

[8] We consider only EC nets whose underlying undirected graph is connected, i.e. nets such that there exists an undirected path between every pair of nodes.

Executability is required because, in the case of strongly-connected EC net fragments, it is possible that successive choice resolutions belong to different branches, and hence accumulate tokens in various branches and cause deadlocks even when each subnet including only one branch by itself does not. An example is shown in Figure 3, where both the subnets including in one case $t_1t_2t_4t_6t_8$, in the other $t_1t_3t_5t_7t_8$, are fireable in isolation. However, when t_2 and t_3 fire in sequence, the complete net deadlocks (one token in p_2 and one token in p_3 are not sufficient to fire any transition).

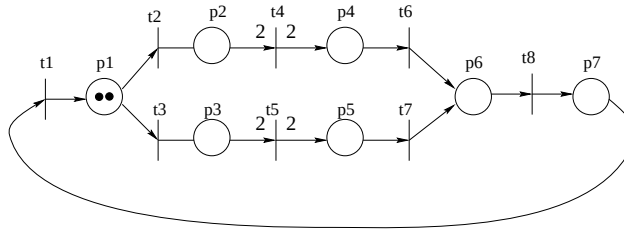


Fig. 3. EC net without executable Σ

We can also consider the scheduling problem as a game played against an adversary who can arbitrarily choose among conflicting transitions and has the goal of accumulating infinitely many tokens in any place of the net. Then, our objective is to find a non-terminating bounded memory execution by matching his choices with a cyclic schedule that returns the net to the initial marking.

Let us consider three examples. Given the net in figure 2a, $\Sigma = \{(t_1t_2t_4), (t_1t_3t_5)\}$ is a valid schedule because it contains a firing sequence for every value of the tokens in place p_1, i.e., whatever conflicting transition fires among t_2 and t_3, it is possible to complete the cycle that returns the net to the initial marking by firing t_4 or t_5 respectively. The net shown in figure 2b is not schedulable because there is no set of finite complete cycles that satisfies Definition 6. For example the set $\Sigma = \{(t_1t_2t_1t_3t_4), (t_1t_3t_1t_2t_4)\}$ is not valid because it does not contain any firing sequence beginning with $t_1t_2t_1t_2$. This corresponds to the fact that, in case transition t_2 (t_3) is always fired, there exists no firing sequence that returns the net to the initial marking and therefore unbounded accumulation of tokens occurs in place p_2 (p_3).

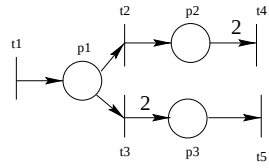


Fig. 4. Schedulable EC net with weighted arcs

The net shown in figure 4 is schedulable and $\Sigma = \{(t_1 t_2 t_1 t_2 t_4), (t_1 t_3 t_5 t_5)\}$ is a valid schedule. The weight two on the input arc of transition t_4 implies that t_2 has to fire twice before transition t_4 is enabled. However, there is no guarantee that this happens within a cycle because it is not possible to know a priori which transition among t_2 and t_3 fires. So, if transitions $t_1 t_2 t_1 t_3 t_5 t_5$ fire in this order, one token remains in place p_2 and the net does not return to the initial marking. The net is considered schedulable because repeated executions of this sequence do not result in unbounded accumulation of tokens (as soon as there are two tokens in place p_2, transition t_4 is fired and the tokens are consumed). The last example shows that a valid schedule does not necessarily include all the possible cyclic firing sequences, some even of infinite length, that can occur depending on the resolution of the non-deterministic choices (in this case the set would be $\{t_1 t_3 t_5 t_5, t_1 t_2 (t_1 t_3 t_5 t_5)^n t_1 t_2 t_4, \forall n \in \mathbb{N} \cup \{\infty\}\}$). A valid schedule should be intended only as a complete set of cyclic firing sequences that ensure bounded memory execution of the net. The set is complete in the sense that it is possible to derive from it a C-code implementation of the schedule including all the sequences that can occur, as we discuss in Section 5.

4.2 How to Find a Valid Schedule

The algorithm is composed of two steps: 1) find a valid set of finite complete cycles, 2) check if the set is executable. Let us consider step 1. To find a valid set of finite complete cycles the net is first decomposed into as many Conflict Free (CF) components as the number of possible solutions for the non-deterministic choices of the net. Then, each component is statically scheduled. If every component is schedulable, we take a set that contains one finite complete cycle for each CF component. If at least one of the CF components is not schedulable, the net itself is said to be not schedulable.

Definition 10. *Let $N = (T, P, F)$ and $N' = (T', P', F')$ be two PNs. N' is a* **subnet** *of N if $T' \subseteq T$, $P' \subseteq P$ and $F' = F \cap ((T' \times P') \cup (P' \times T'))$.*

Definition 11. *A subnet N' is a* **transition-generated subnet** *of N if P' is the set of all predecessors and successors places in N of the transitions in T' (i.e. $t \in T' \Rightarrow t^\bullet \subseteq P', {}^\bullet t \subseteq P'$).*

Definition 12. *A* **T-component** *N' of a net N is a set of transition-generated subnets such that each of them is a consistent Conflict Free net and for each initially enabled transition t in N', there exists a T-invariant containing t.*

The net in figure 5 is not a T-component: it is consistent because there exists a T-invariant ($f(\sigma) = (0, 0, 1, 1)$), but there is no T-invariant containing transition t_1. This corresponds to unbounded accumulation of tokens produced by the source transition t_1 occurring in place p_1.

Definition 13. *A T-component N' is (statically)* **schedulable** *if there exists a firing sequence that returns it to the initial marking without any deadlock when its execution is simulated.*

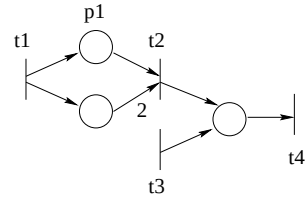


Fig. 5. Conflict Free net

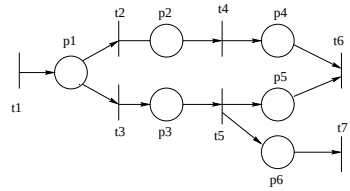


Fig. 6. Not schedulable EC net

Definition 14. *A* **T-allocation** *over an EC net is a function* $\alpha : P \rightarrow T$ *that chooses exactly one among the successors of each place* $(\forall p \in P, \alpha(p) \in p^\bullet)$.

Definition 15. *The* **T-reduction** *associated with a T-allocation is a set of subnets generated from the image of the T-allocation using the following Reduction Algorithm.*

Let C_1,C_2, ...C_M be the Equal Conflict Sets defined in Section 2.3 as the equivalence classes of the Equal Conflict Relation and α_i the i-th T-allocation containing at most one transition for every C_j. The T-reduction $R_i = (T_{R_i}, P_{R_i}, F_{R_i})$ corresponding to T-allocation α_i is generated as follows (see figure 7).
Reduction Algorithm (Modified from [9])

1. $R_i = N$ $(T_{R_i} = T,\ P_{R_i} = P,\ F_{R_i} = F)$.
2. For all $t_k \in T_{R_i}$ and $t_k \notin \alpha_i$
   (a) Remove t_k.
   (b) $\forall s \in t_k^\bullet$, remove place s unless one of the following conditions holds:
      i. s has a predecessor transition different from t_k $(\exists t \in^\bullet s$ s.t. $t \in T_{R_i})$.
      ii. the successor transition of s has a predecessor place that is different from s and is not a source place $(\exists t \in^\bullet ({}^\bullet(s^\bullet))$ s.t. $t \in T_{R_i})$.
   (c) If s_i is a removed place, $\forall t_j \in s_i^\bullet$, remove t_j if one of the following conditions holds:
      i. t_j has no predecessor place $(|{}^\bullet t_j| = 0)$.
      ii. all predecessors of t_j are source places. In this case remove every $s \in^\bullet t_j$.
   (d) Apply the previous two steps until they cannot be applied any longer.

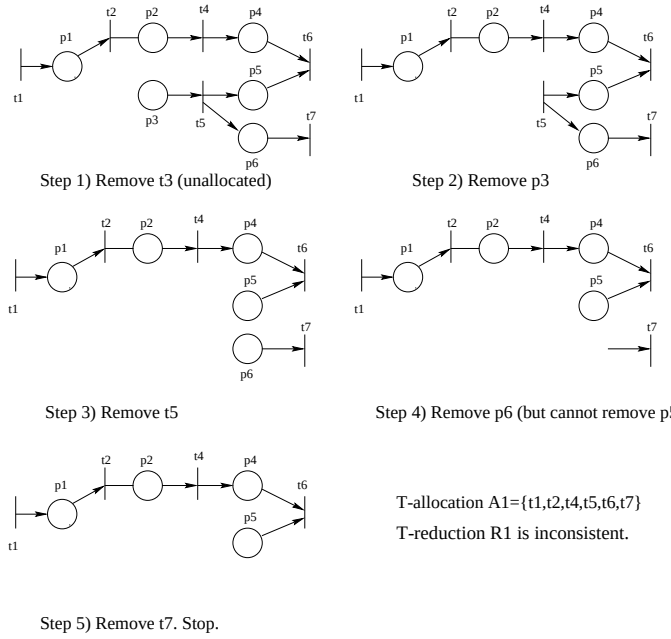


Fig. 7. How to obtain a T-reduction from the net shown in figure 6

Intuitively, the algorithm removes the part of the net that is inactive when the conflicting transitions included in the T-allocation are always chosen. Inactivity should be interpreted for a transition as the capability of firing only a finite number of times [9]. For places inactivity means having empty presets in the current net. Let us consider the EC net in figure 8 and apply the Reduction algorithm to compute the T-reduction from the T-allocation A_1 in figure 9. First, unallocated transitions (t_3 and t_6) are removed. Then, successor places of every removed transition (in this case p_3) are deleted unless they are merge places with undeleted predecessor transitions or their successor is a transition feeded by at least one active places. The next step consists of the elimination of the transitions that are successors of places deleted in the previous step. At this point, transitions are deleted only if they have no predecessor places (so we can remove transition t_{10}) or all the predecessor places are inactive. The last two steps are iterated until no new node is deleted. In this example, the algorithm does not continue because place p_8 cannot be removed since it has a predecessor transition (t_{11}). When the final T-reduction obtained by applying this algorithm contains one or more inactive places, the net is not schedulable, since these places contain only a finite number of tokens and do not allow infinite execution.

Theorem 16. *The T-reduction R_i obtained by applying the reduction algorithm is*

1. *a set of transition-generated Conflict Free nets $\{R_i^1, R_i^2, ...R_i^M\}$.*

2. *a T-component of N iff every subnet R_i^l is consistent and for each initially enabled transition $t \in R_i$ there exists a T-invariant containing t.*

Proof. 1. Each R_i is a Conflict Free net by construction because it contains at most one transition for every Equal Conflict Set. Let's show that each subnet obtained from the Reduction Algorithm is transition-generated. We need to prove that $\forall t_k \in R_{T_i}$ and $\forall s \in {}^\bullet t_k \cup t_k^\bullet$, $s \in R_{P_i}$. Consider places $s \in t_k^\bullet$, the algorithm does not remove a place s if $\exists t \in {}^\bullet s$ s.t. $t \in R_{T_i}$ (Reduction Algorithm (b)i). For places $s \in^\bullet t_k$, s is removed only if all other predecessor places of t_k are source places (Reduction Algorithm (b)ii). In this case at the next iteration step also t_k is removed (Reduction Algorithm (c)ii). Therefore, if $t \in R_{T_i}$ ${t^\bullet} \cup^\bullet t \subseteq R_{P_i}$.
2. Both directions trivially follow from part 1 of Theorem 16 and Definition 12.

Definition 17. *An EC net N is* **T-allocatable** *if every T-reduction generated from a T-allocation is a T-component.*

Theorem 18. *A set of finite complete cycles Σ that does not contain at least one finite complete cycle for every T-reduction is not valid.*

Proof. By contradiction. Assume that a set of finite complete cycles Σ not including any finite complete cycle for a T-reduction and including one finite complete cycle for any other T-reduction is valid. Consider two T-reductions: R_j derived from T-allocation A_j and R_i derived from A_i. R_i and R_j contain the same set of allocated transitions except for transitions t_m and t_n, such that $\Theta(t_m, t_n) = 1$ and $t_m \in R_i$ and $t_n \in R_j$. Since Σ is a valid set and includes one finite complete cycle σ_j for R_j, by Definition 6 it includes a sequence σ_x that is equal to σ_j upto transition t_n and has transition t_m instead of t_n, while all the other allocated transitions that appear in σ_j appear also in σ_x. Sequence σ_x can be a finite complete cycle only for R_i since no other T-reduction contains the same set of allocated transitions as R_j (except t_m). This contradicts the hypothesis that Σ contains no finite complete cycle for T-reduction R_i. Therefore, the assumption that Σ is valid was not true.

The following fundamental theorem states that T-allocatability, together with schedulability of every T-component, is a necessary and sufficient condition for the existence of a valid set of finite complete cycles. A net is T-allocatable if for all its components, each of them corresponding to a sequence of choices, there exists a cyclic schedule containing at least one occurrence of every initially enabled transition of the component. This means that a T-allocatable net, if there is no deadlock during execution of the cyclic schedules, can be executed forever with bounded memory, because for every choice there is always the possibility to complete successfully a finite cycle of firings that returns the net to the initial marking. Intuitively, T-allocations can be interpreted as control functions that choose which transition fires among several conflicting ones and therefore which component of the net is active at every cycle.

Theorem 19. *Given a EC net, there exists a valid set of finite complete cycles iff it is T-allocatable and every T-component is schedulable.*

Proof. ($\Rightarrow$) Let us prove that if the net is not T-allocatable or there exists a T-component that is not schedulable, there exists no valid set of finite complete cycles.

- Case 1: the net is not T-allocatable. This means that there exists at least one T-allocation A_i for which the generated T-reduction R_i is not a T-component. This may occur either if R_i is not consistent or if there exists at least an initially enabled transition $t_k \in R_i$ such that no T-invariant of R_i contains t_k. In the first case, i.e. R_i is not consistent, there is no finite complete cycle for the T-reduction R_i (i.e. for some non-deterministic choice resolutions there is unbounded accumulation of tokens). In the second case, i.e. there exists a finite complete cycle fcc_i for R_i but not including transition t_k, this finite complete cycle cannot be included in the valid set Σ because it does not satisfy the fairness condition of Definition 6. In both cases, any set Σ of finite complete cycles does not contain a finite complete cycle for T-Reduction R_i. For Theorem 18, Σ is not a valid set of finite complete cycles.
- Case 2. If a T-component R_i is not schedulable, there is no finite complete cycle for R_i and using the same argument above there exists no valid set of finite complete cycles.

($\Leftarrow$) By construction. The net is T-allocatable and each T-component is a schedulable Conflict Free net. We derive a valid set of finite complete cycles Σ, as follows: first of all, consider an arbitrarily chosen T-component TC_i and insert in Σ a finite complete cycle σ_i containing at least one occurrence of every transition in TC_i . Then, iterate for all sequences in Σ the following step: for every first occurrence of each conflicting transition $t_m \in \sigma_i$ insert in Σ a finite complete cycle σ_j for the T-component TC_j which contains the same allocated transitions as TC_i except for t_n instead of t_m. Choose σ_j so that it begins with the same subsequence as σ_i upto the different allocated transition where it contains t_n instead of t_m. Also, σ_j must contain at least one occurrence of every transition in TC_j. The set Σ derived following the above rules satisfies Definition 6. Condition (1) is trivially satisfied by construction. Condition (2) holds because σ_j contains at least one occurrence of every transition of TC_j and, since all the transitions of the net not in TC_j were removed in the Reduction process, they cannot be enabled for all markings reached by σ_j.

To check if a given net is T-allocatable, it is necessary to verify that every T-reduction obtained from the Reduction Algorithm is a T-component. For this purpose one must solve the state equations for every subnet of the T-reduction and check consistency; in the case any subnet of the T-reduction contains merge places, it is also necessary to check for this subnet that every transition initially enabled is in the support of at least one T-invariant. To detect if a T-component is schedulable the simulation based technique for WT-Systems described in Section 3 is used. At this point, finite complete cycles are derived from the T-invariants.

In term of complexity, the number of T-reductions (and therefore the number of applications of the Reduction Algorithm that computes them) is exponential in the number of conflicting transitions. However, simple heuristics can be used

to reduce the cost of the algorithm in most practical cases [10]. The cost of computing a static schedule for each T-reduction is polynomial [2].

Let us describe two examples.

The EC net presented in figure 6 is not T-allocatable. Figure 7 shows the steps of the Reduction Algorithm applied to T-allocation $A_1 = \{t_1, t_2, t_4, t_5, t_6, t_7\}$. The generated T-reduction R_1, that is inconsistent because it contains an inactive place, is not a T-component and the net is not T-allocatable. Therefore, there exists no valid schedule for this EC net. In fact, if sequence $\sigma = (t_1 t_2 t_4)$ is fired infinitely often, there is unbounded accumulation of tokens in place p_4.

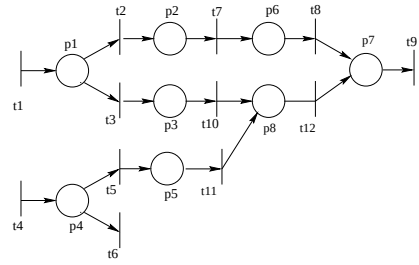


Fig. 8. Schedulable EC net

The EC net shown in figure 8 is T-allocatable. The T-components corresponding to the four T-allocations are represented in figure 9. To find a valid set of finite complete cycles we solve the state equations for each T-reduction. Let us consider T-Reduction R_1. The T-invariants of R_1 are $(1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0)$ and $(0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1)$. From the T-invariants a finite complete cycle is derived only after simulation checks that there is no deadlock. The same procedure is repeated for each T-reduction and a valid set of finite complete cycles for this EC net is $\{(t_1 t_2 t_7 t_8 t_9 t_4 t_5 t_{11} t_{12} t_9), (t_1 t_3 t_{10} t_{12} t_9 t_4 t_5 t_{11} t_{12} t_9), (t_1 t_2 t_7 t_8 t_9 t_4 t_6)$, $(t_1 t_3 t_{10} t_{12} t_9 t_4 t_6)\}$.

Once a valid set of finite complete cycles is obtained, we need to check executability, i.e. whether there are enough tokens at the initial marking to execute all the finite complete cycles without a deadlock. The existence of a valid set of finite complete cycles guarantees by definition that each finite complete cycle is executable in the subnet in which the finite complete cycle was found. However, if the EC net contains a strongly connected subnet with a choice (figure 3), we must conduct an executability check for each of such structures since executability of each component does not imply executability of the whole net.

Consider the net shown in Figure 3. Two T-invariants are obtained as $(t_2 t_2 t_4 t_6$ $t_6 t_8 t_8 t_1 t_1)$ and $(t_3 t_3 t_5 t_7 t_7 t_8 t_8 t_1 t_1)$, where each is defined for a subnet corresponding to a simple cycle of the figure. When the executability of each T-invariant is checked for its subnet, we conduct a symbolic simulation. We assign a variable for each of the places in the subnet that are choices in the original EC net and have initial tokens, e.g. a variable $p_1^{(1)}$ at the place p_1 in the subnet for the first

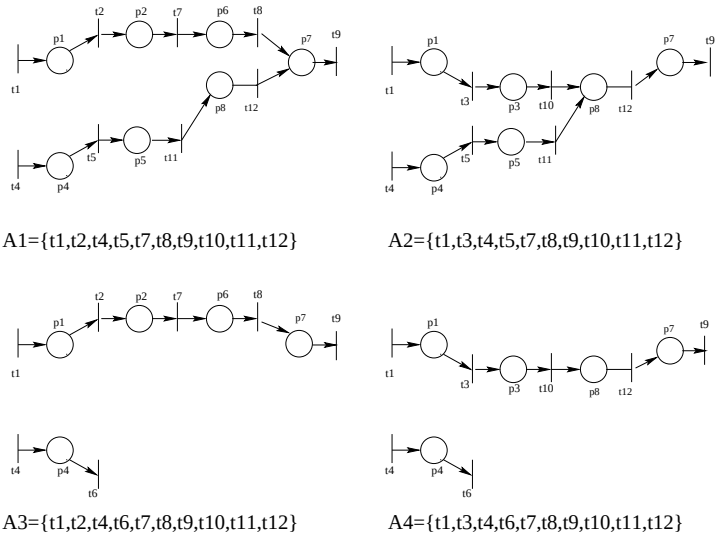


Fig. 9. T-components

T-invariant above (upper cycle in the figure). We then simulate the T-invariant, using these variables as the initial numbers of tokens at such places while we use the actual number of tokens for a place that is initially marked but no variable is defined for. The result of the simulation is an inequality of the variables such that given initial numbers of tokens allow the T-invariant to be executed if and only if the inequality holds. For our example, we obtain $p_1^{(1)} \geq 2$, which means that the place p_1 must have at least two tokens in order to execute the first T-invariant in its subnet. Similarly, for the second T-invariant, we obtain $p_1^{(2)} \geq 2$, where $p_1^{(2)}$ is the variable defined at p_1 for the corresponding subnet. Since p_1 is initially marked with two tokens, we also have an equation $p_1^{(1)} + p_1^{(2)} = 2$. Now, a value of each variable represents the number of tokens at the place that can be used to fire transitions in the corresponding subnet. Thus Σ is executable if for every integer assignment of these variables which satisfies the equation, at least one of the inequalities holds. Equivalently, Σ is *not* executable if there exists an integer solution which satisfies the equation but none of the inequalities. In this example, by reversing the directions of the inequalities, our problem is to find a solution for the system made of $p_1^{(1)} + p_1^{(2)} = 2$, $p_1^{(1)} < 2$, and $p_1^{(2)} < 2$. Since $p_1^{(1)} = p_1^{(2)} = 1$ is such a solution, we conclude that this valid set of finite complete cycles is not executable. In general, we obtain a set of inequalities and equalities, where each inequality is a boolean disjunction of linear equations and each equality is a linear equation. The problem can be solved by iteratively solving an integer programming. Alternatively, the equalities and inequalities may be represented using decision diagrams, e.g. [5], and the existence of a solution can be found with a constant time by tautology checking.

4.3 BDF and Petri Nets

The problem of scheduling specifications containing data-dependent control was first addressed in [7]. Buck and Lee [7] introduced the Boolean Data Flow (BDF) networks model and proposed an algorithm to compute a quasi-static schedule. However, the problem of scheduling BDFs with bounded memory is undecidable, i.e. any algorithm may fail to find a schedule even if the BDF is schedulable. Hence, the algorithm proposed by Buck can find a solution only in special cases. On the contrary, schedulability of PNs is decidable as a consequence of the decidability of the reachability problem [11]. PNs and Boolean Dataflow networks (BDF) [7], both used to model specifications containing data-dependent control, have different expressive power. One difference is in the semantics of the communication channels among blocks. Channels in BDFs are FIFO queues that preserve the order of the valued tokens. Instead, PNs do not have FIFO semantics because the tokens do not carry values [7]. This implies also that in the PNs model, differently from BDF, there can be no correlation among the results of different non-deterministic choices. The latter property in particular implies that some schedulable BDF nets are not schedulable when modeled as an EC net, just as some schedulable BDFs are not schedulable according to [7].

BDF is a determinate model, since all the valid executions of the network produce the same token streams regardless of the order in which the actors are executed [7], while PNs are not determinate when they contain merge places. However, it is possible to make a PN determinate by imposing appropriate conditions to the schedule. For example, the net shown in figure 10, is not determinate

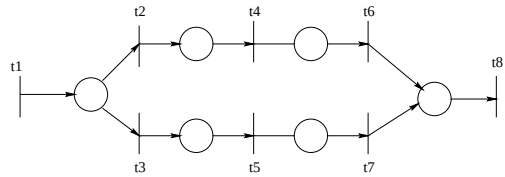


Fig. 10. Example of non-determinate PN

because of the presence of the non-deterministic merge, i.e. there is no guarantee on the order of the tokens in the output stream if t_6 and t_7 are concurrently enabled. To enforce determinacy, it is necessary to impose some restrictions to the scheduler and ensure that the states where two merging transitions are concurrently enabled are never reached. One way is to allow tokens to enter the net (i.e. firing t_1) only if no other transition is enabled. Further investigation in this direction is necessary to formally define and prove the conditions under which a PN is determinate.

5 C-Code Generation

The final goal of this paper is the synthesis of a software implementation [9] that satisfies functional correctness and minimizes a cost function of latency and memory size. In general, such implementation consists of a set of software tasks that are invoked by the Real Time Operating System either by interrupt or polling. Our approach for software synthesis derives an implementation directly from a valid schedule, that should be intended as an intermediate description, containing in explicit form a set of rules, such as number and order of firing of transitions, that an implementation should follow to guarantee bounded memory execution. In this Section we show how to generate from a valid schedule an implementation that consists of as many fragments of C code (*tasks*) as the number of source transitions with independent firing rate. Transitions with independent firing rate cannot be quasi-statically scheduled together and therefore a task is composed only of transitions with dependent firing rates, that are transitions belonging to the same T-invariant.

Schedule (Σ)
while ($t_i \neq EOS$) { Task(Σ,i); $i = i + 1$; }

Task (Σ,*i*)
while ($t_i \neq EOT$) { if t_i is already visited
  then{ insert **goto** label t_i }
  else{
    if t_i is a conflicting transition { insert **if..then..else** }
      if $f(t_i) < f(t_{i-1})$ { insert counting var and **if** test }
      if $f(t_i) > f(t_{i-1})$ { insert counting var and **while** test }
      if $f(t_i) = f(t_{i-1})$ { insert t_i } } }
end Task

Fig. 11. Pseudo-code of the code generation algorithm

A pseudocode description of the algorithm that generates C code is shown in figure 11 (EOS means End of Sequence and EOT means End of T-Invariant). The routine *Schedule* visits all the transitions in the valid schedule Σ, by calling the routine Task every time a new T-invariant is visited. *Task* checks if a transition has already been visited and, if so, inserts a label and a goto to avoid repetition of code. This corresponds to the presence of a merge place in the EC net model that yields code patterns which are common either to the branches of an *if-then-else* or are shared by different tasks. Instead, if the transition currently visited is a conflicting one, an if-then-else structure is generated and the code in the two branches is synthesized by traversing the two finite complete cycles of Σ containing the conflicting transitions. In case of multirate nets, a variable counting the

[9] We consider uni-processor architecures, that are used for most embeeded systems applications like cellular phones, multimedia set-top boxes etc.

number of tokens and a test are used to determine whether an operation should be executed. Figure 12 provides an example of C program generated for the net shown in figure 4 and whose valid schedule is $\Sigma = \{(t_1 t_2 t_1 t_2 t_4)(t_1 t_3 t_5 t_5)\}$.

```
while (true) {
   t1;
   if (p1) {
      t2; count(p2)++;
      if (count(p2) == 2) {
            t4; count(p2)- =2; }
         } else {
         t3; count(p3)+=2;
         while (count(p3 ≥ 1) {
            t5; count(p3)- -; } } }
```

Fig. 12. C code for the EC net of figure 4

6 Experimental Results

We applied our algorithm to synthesize a software implementation of a real life embedded system: an ATM server for Virtual Private Networks [4]. The main functionalities of the server are (1) a message discarding technique (MSD) that avoids node congestion and (2) a bandwidth control policy based on a Weighted Fair Queueing (WFQ) scheduling discipline. Figure 13a gives a high-level description of the algorithm. The inputs of the system are *Cell*, an interrupt that occurs at irregular times when a non-empty cell enters the Server and *Tick*, an event that periodically triggers the process of forwarding the next outgoing cell to the output port. Therefore, Cell and Tick are inputs with independent firing rate. The module MSD decides whether an incoming cell must be accepted and the module CELL_EXTRACT selects, every cell slot, which cell must be emitted among those stored in the internal buffer. WFQ_SCHEDULING may be activated either by MSD or by CELL_EXTRACT and computes the cell emission time. We have chosen this example because it implements a data-dominated algorithm containing several data-dependent control structures. We modeled the algorithm using a ECN containing 49 transitions and 41 places (figure 13b), of which 11 non-deterministic choices. From the ECN model we could derive a valid schedule containing 120 finite complete cycles, one for each different T-reduction, and from the valid schedule we obtained a software implementation composed of two tasks, one for each input with independent firing rate.

In table I we compare two software implementations: the first, named QSS, was obtained using our Quasi-Static Scheduling based technique and consists of two tasks, the second, named functional task partitioning, consists of five tasks and was obtained by synthesizing separately one tasks for each of the five modules shown in figure 13. The results, obtained using a testbench of 50 ATM

Sw implementation	QSS	Functional task partitioning
Number of tasks	2	5
Lines of C code	1664	2187
Clock cycles	197526	249726

Table I

cells, show that the number of clock cycles and the code size are significantly smaller for the QSS implementation that is composed of a smaller number of tasks and therefore has a smaller overhead due to tasks activation.

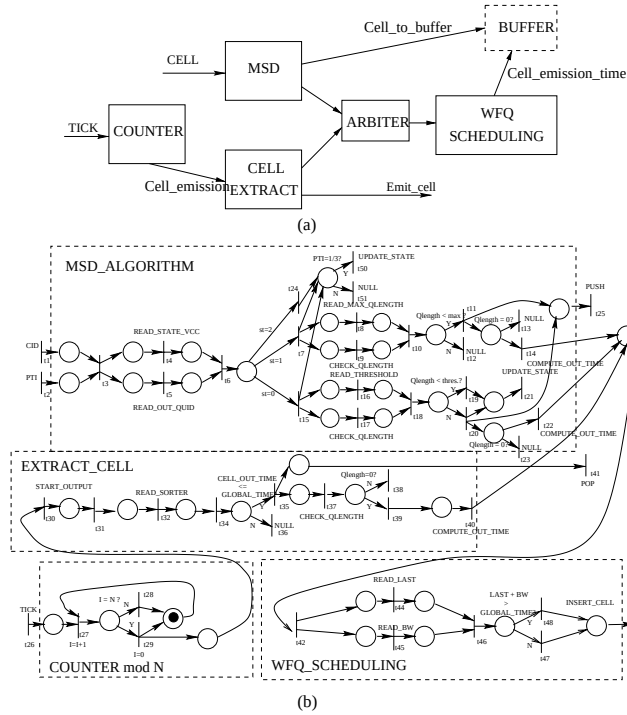


Fig. 13. ATM Server functional description (a) and ECN model (b)

7 Conclusions

In this paper we have defined schedulability for Equal Conflict Nets and presented a quasi-static scheduling (QSS) algorithm that finds a schedule, whenever there exists one. This result is important because QSS, maximizing the amount of work done at compile time, allows to reduce significantly the run-time scheduling overhead. Moreover, we have explained why EC nets are an appropriate

model for embedded systems specifications containing data processing and data-dependent control, and compared them with other models. We have showed how a valid schedule can be found by extending well-established techniques used for scheduling Weighted T-Systems, and presented also how to generate a C code implementation of the schedule. The presented algorithms have been applied to a real case study, an ATM Server, and their potential has been demonstrated.

Acknowledgments

The authors want to thank Jordi Cortadella, Mike Kishinewsky, Alex Kondratyev, Howard Wong-Toi and Lothar Thiele for useful comments and discussions.

References

1. J. Desel and J.Esparza. *Free choice Petri nets.* Cambridge University Press, 1995.
2. E.A.Lee and D.G.Messerschmitt. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Transactions on computers*, January 1987.
3. E.Best. Structure theory of petri nets: the free choice hiatus. In *Advances in Petri Nets*, 1986.
4. E.Filippi et al. Intellectual property re-use in embedded system co-design: an industrial case study. In *International Symposium System Synthesis. ISSS '98. Taiwan*, December 1998.
5. I.R.Bahar et. al. Algebraic decision diagrams and their applications. In *IEEE International Conference on Computer-Aided Design*, November 1993.
6. E.Teruel and M.Silva. Structure theory of equal conflict systems. In *Theoretical Computer Science, vol.153, pp. 271-300*, 1996.
7. J.Buck. *Scheduling dynamic dataflow graphs with bounded memory using the token flow model.* Ph.D dissertation. UC Berkeley, 1993.
8. B. Lin. Software synthesis of process-based concurrent programs. In *Proceedings of the Design Automation Conference*, June 1998.
9. M.Hack. *Analysis of Production Schemata by Petri Nets.* Master thesis. MIT, 1972.
10. M. Sgroi. *Quasi-Static Scheduling of Embedded Software Using Free-Choice Petri Nets.* M.S. dissertation. UC Berkeley, May 1998.
11. T.Murata. Petri nets: properties, analysis and applications. In *Proceedings of the IEEE*, April 1989.

Theoretical Aspects of Recursive Petri Nets

Serge Haddad[1] and Denis Poitrenaud[2]

[1] LAMSADE - UPRESA 7024, Université Paris IX, Dauphine
Place du Maréchal Delattre de Tassigny, 75775 Paris cedex 16
[2] LIP6 - UMR 7606, Université Paris VI, Jussieu
4, Place Jussieu, 75252 Paris cedex 05

Abstract. The model of recursive Petri nets (RPNs) has been introduced in the field of multi-agent systems in order to model flexible plans for agents. In this paper we focus on some theoretical aspects of RPNs. More precisely, we show that this model is a strict extension of the model of Petri nets in the following sense : the family of languages of RPNs strictly includes the union of Petri net and Context Free languages. Then we prove the main result of this work, the decidability of the reachability problem for RPNs.

1 Introduction

Since the introduction of Petri nets, even before the decidability of the reachability problem has been solved, theoretical works have been developed in order to study the impact of extensions of Petri nets on this problem. For instance, the reachability problem is undecidable for Petri nets with two inhibitor arcs while it becomes decidable with one inhibitor arc or a nested structure of inhibitor arcs [Rei95]. The self-modifying nets introduced by R. Valk have (like Petri nets with inhibitor arcs) the power of Turing machine and thus many properties including reachability are undecidable [Val78a,Val78b]. Introducing restrictions on self-modifying nets enables to decide some properties [DFS98] (boundedness, coverability, termination,...) but the reachability remains undecidable.

Recently Recursive Petri nets (RPNs) have been proposed for modeling plans of agents in a multi-agent system [SH95,SH96]. A RPN has the same structure as an ordinary one except that the transitions are partitioned into three categories : elementary transitions, abstract transitions and final transitions. Moreover a starting marking is associated to each abstract transition. The semantics of such a net may be informally explained as follows. In an ordinary net, a thread plays the token game by firing a transition and updating the current marking (its internal state). In a RPN there is a dynamical tree of threads (denoting the fatherhood relation) where each thread plays its own token game. The step of a RPN is thus a step of one of its threads. If the thread fires an elementary transition, then it updates its current marking using the ordinary firing rule. If the thread fires an abstract transition, it consumes the input tokens of the transition and generates a new child which begins its token game with the starting marking of the transition. If the thread fires a final transition, it aborts its whole descent

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 228–247, 1999.
© Springer-Verlag Berlin Heidelberg 1999

of threads, produces (in the token game of its father) the output tokens of the abstract transition which gave birth to it and dies. In case of the root thread, one obtains an empty tree.

The modeling capabilities of RPNs can be illustrated in different ways. In a subsequent section, we present the modeling of faults within a system whereas an equivalent modeling by an ordinary Petri net is not known to be possible. RPNs enable also to easily model multi-level executions (e.g. interruptions). In case of planning of agents, abstract transitions model differed plannings of complex actions.

As soon as a new model is proposed, a significant question (at least if a semantics of the firing sequence is possible) is to know whether the model is really an extension or simply an abbreviation. For instance the model of colored Petri nets with finite color domains is an abbreviation whereas allowing infinite domains extends the ordinary Petri nets. In case of RPNs we show that this model is a strict extension of ordinary Petri nets. Our proof is based on a result in [Jan79] which establishes that the palindrome language is not a Petri net language (even in the largest definitions). Thus we exhibit a RPN which recognizes this language. Moreover, we prove that RPN languages strictly include the union of Context Free and Petri net languages.

Then we tackle with the main question about strict extensions of Petri nets: does the reachability problem remain decidable ? This question is theoretically important as it seems a limit result. Recently in [Rei95], it has been stated that reachability is decidable for Petri nets with one inhibitor arc (or more generally with a nested structure of inhibitor arcs). We prove that the reachability problem is also decidable for RPNs. Our proof is divided into three steps. First we prove that one can decide whether the thread initiated by an abstract transition can die by some sequence in which case we call such a transition a "closable transition". Then we study the structure of a hypothetical sequence for the reachability and show that its existence is equivalent to the existence of sequences of reachability for initial and final states of less complexity (in terms of the size of their tree of threads). Finally we show that when the initial and final states are associated to one (or zero) thread, then the problem is equivalent to the ordinary reachability where the non closable abstract transitions are deleted and each closable one is replaced by an equivalent elementary one.

The balance of the paper is the following. In Sect. 2, we define the RPNs, we give some examples in order to understand their behavior and their modeling capability. Then, we show that the model is a strict extension of Petri nets and that their languages strictly include the union of Petri net and Context Free languages. In Sect. 3, we prove the decidability of the reachability problem. In the last section, we conclude giving some perspectives to this work.

2 Recursive Petri Nets

2.1 Structure

A *recursive Petri net* is defined by a tuple $N = \langle P, T, W^-, W^+, \Omega \rangle$ where

- P is a finite set of places.
- T is a finite set of transitions.
- A transition of T can be either elementary, abstract or final. The sets of elementary, abstract and final transitions are respectively denoted by T_{el}, T_{ab} and T_{fi} (with $T = T_{el} \uplus T_{ab} \uplus T_{fi}$ where $\uplus$ denotes the disjoint union).
- W^- and W^+ are the pre and post flow functions defined from $P \times T$ to $\mathbb{N}$.
- Ω is a labeling function which associates to each abstract transition an ordinary marking (i.e. an element of $\mathbb{N}^P$).

An *extended marking* tr of a recursive Petri net $N = \langle P, T, W^-, W^+, \Omega \rangle$ is a labeled tree $tr = \langle V, M, E, A \rangle$ where V is the set of vertices, M is a mapping $V \to \mathbb{N}^P$, $E \subseteq V \times V$ is the set of edges and A is a mapping $E \to T_{ab}$. We denote by $v_0(tr)$ the root node of the extended marking tr. The edges E build a tree i.e. for each v different from $v_0(tr)$ there is one and only one $(v', v) \in E$ and there is no $(v, v_0(tr)) \in E$. Any ordinary marking can be seen as an extended marking composed by a unique node. The empty tree is denoted by $\perp$.

Remark: An extended marking does not depend on V the set of vertices. Given two extended markings, if there is a one-to-one mapping between the two sets of markings which preserves the set of edges, the labeling of vertices and of the edges then the two markings are equal.

A *marked recursive Petri net* (N, tr_0) is a recursive net N associated to an initial extended marking tr_0. This initial extended marking is usually a tree reduced to a unique vertex.

For a vertex v of an extended marking, we denote by $pred(v)$ its (unique) predecessor in the tree (defined only if v is different from the root) and by $Succ(v)$ the set of its direct and indirect successors including v ($\forall v \in V, Succ(v) = \{v' \in V \mid (v, v') \in E^*\}$ where E^* denotes the reflexive and transitive closure of E).

A *branch* br of an extended marking tr is one of the subtrees rooted at a son of $v_0(tr)$. One can associate to a branch a couple (t, tr) where t is the abstract transition which labels the edge leading to the subtree and tr the subtree taken in isolation. Let us note that the couple (t, tr) characterizes a branch.

In other words, given an extended marking tr, a branch br with its couple (t, tr') fulfills : tr' is a sub-tree of tr verifying $(v_0(tr), v_0(tr')) \in E$ (i.e. in tr, the root of tr' is a direct successor of the root of tr) and $A(v_0(tr), v_0(tr')) = t$ (i.e. in tr, the arc between the root of tr and tr' is labeled by t).

We denote by $Branch(tr)$ the set of branches of an extended marking tr and by $branch(tr, t)$ the subset of Branch(tr) where the edge leading to the subtree is labeled by t.

We denote by (m, Br), where m is an ordinary marking and Br a set of branches, the extended marking tr verifying $M(v_0(tr)) = m \wedge Branch(tr) = Br$.

The depth of an extended marking is recursively defined as follows : let m be an ordinary marking and tr an extended marking.

- $depth(\bot) = 0$
- $depth(m) = 1$
- $depth(tr) = max(\{depth(tr') \mid \exists (t, tr') \in Branch(tr)\}) + 1$

2.2 Semantics

A transition t is enabled in a vertex v of an extended marking tr iff $\forall p \in P, M(v)(p) \geq W^-(p,t)$. In other words, the thread associated to each node uses the same rule for the enabling of a transition as for ordinary Petri nets.

We denote by $tr \xrightarrow{t}$ that there exists a node v of tr in which the transition t is firable.

The firing of an enabled transition t from a vertex v of an extended marking $tr = \langle V, M, E, A \rangle$ leads to the extended marking $tr' = \langle V', M', E', A' \rangle$ depending on the type of t:

***t* is an *elementary transition* ($t \in T_{el}$).** The thread associated to v fires such a transition as for ordinary Petri nets. The structure of the tree is unchanged. Only the current marking of v is updated.

- $V' = V$
- $\forall v' \in V \setminus \{v\}, M'(v') = M(v')$
  $\forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t) + W^+(p,t)$
- $E' = E$
- $\forall e \in E, A'(e) = A(e)$

***t* is an *abstract transition* ($t \in T_{ab}$).** The thread associated to v consumes the input tokens of t. It generates a new thread v' with initial marking the starting marking of t. Let us note that the identifier v' is a fresh identifier absent in V.

- $V' = V \cup \{v'\}$
- $\forall v'' \in V \setminus \{v\}, M'(v'') = M(v'')$
  $\forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t)$
  $M'(v') = \Omega(t)$
- $E' = E \cup \{(v, v')\}$
- $\forall e \in E, A'(e) = A(e)$
  $A'((v, v')) = t$

***t* Is a *Final Transition* ($t \in T_{fi}$).** If the thread is associated to the root of the tree, the firing leads to the empty tree. In the other case, the thread associated to v produces the output tokens of the abstract transition which gave birth to it, in the marking of its father Then it (and its whole descent) dies.

- $V' = V \setminus Succ(v)$
- $\forall v' \in V' \setminus \{pred(v)\}, M'(v') = M(v')$
  $\forall p \in P, M'(pred(v))(p) = M(pred(v))(p) + W^+(p, A(pred(v), v))$
- $E' = E \cap (V' \times V')$
- $\forall e \in E', A'(e) = A(e)$

We denote by $tr \xrightarrow{t} tr'$ that there exists a node v of tr such that the firing of t in v leads the net to the extended marking tr'.

A firing sequence is usually defined : a transition sequence $\sigma = t_0 t_1 t_2 \dots t_n$ is enabled from an extended marking tr_0 (denoted by $tr_0 \xrightarrow{\sigma}$) iff there exists tr_1, tr_2, ..., tr_n such that $tr_{i-1} \xrightarrow{t_i} tr_i$ for $i \in [1, n]$.

Important remark: In a firing sequence, we impose (w.l.o.g.) that any fresh identifier is new not only in the current tree but also in all the previous trees of the sequence. Such a restriction ensures that if the roots of two branches of two trees of the sequence are associated to the same identifier then they denote the same branch - with possible change of structure and marking.

We denote by $\mathcal{L}(N, tr_0, Tr_f)$ (where Tr_f is a finite marking set) the set of firing sequences of N from tr_0 to a marking of Tr_f. This set is called the language of N.

Let σ be a firing sequence and tr_1, tr_2, ..., tr_n the extended markings visited by σ, the depth of σ (denoted by $Depth(\sigma)$) is the maximal depth of tr_1, tr_2, ..., tr_n.

2.3 Recursive Petri Nets versus Petri Nets

In this section, we show that the recursive Petri net model is a strict extension of Petri nets.

For this demonstration, we consider *labeled recursive Petri nets*. Such a net associates to a recursive net, a labeling function h defined from the transition set T to an alphabet Σ plus λ (the empty word). h is extended to sequences and also to languages. The language of a labeled recursive Petri net is defined by $h(\mathcal{L}(N, tr_0, Tr_f))$.

The Fig. 1 gives a marked labeled recursive Petri net. The marking associated to both abstract transitions is P.

Figure 2 presents a prefix of its reachability graph. Notice that the complete graph is infinite. From the initial marking, all the transitions adjacent to the place P are enabled. The firing of the abstract transition labeled a consumes the token of the place P and leads to the construction of a successor node associated to the marking P. The arc between the two nodes is labeled by an a. The firing of the final transition labeled a from the initial marking leads to the empty tree.

M. Jantzen has shown in [Jan79] (see also [Lam88]) that $PAL(\Sigma)$ (for $|\Sigma| \geq 2$) is not a Petri net language (allowing λ-transitions).

We consider the language of the recursive net of the Fig. 1 defined by the set of ending markings $\{\perp, P\}$ (the two corresponding markings are represented

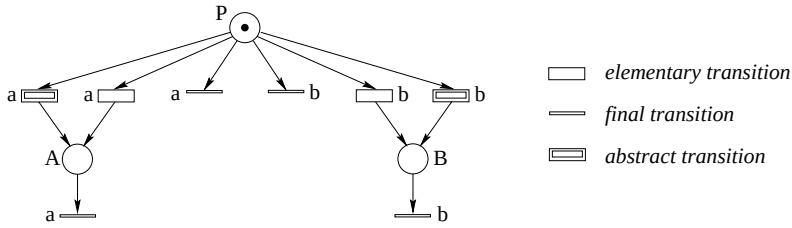


Fig. 1. recursive Petri net which generates palindromes on $\{a, b\}$

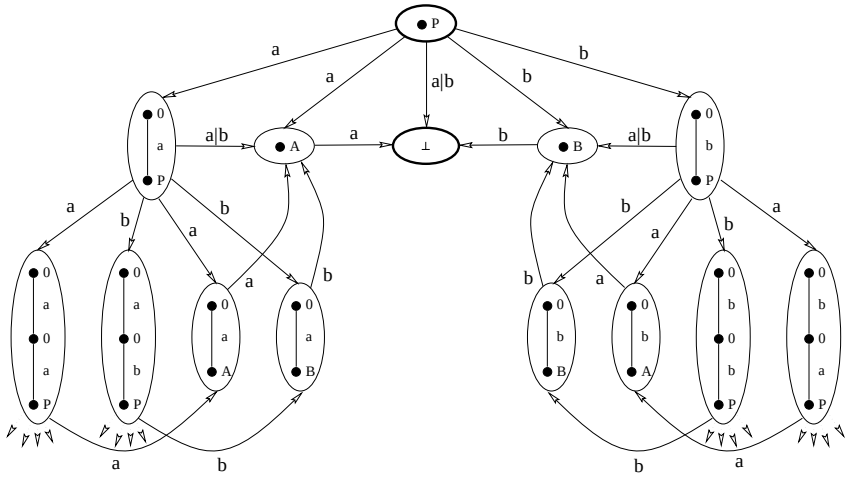


Fig. 2. a prefix of its reachability graph

in bold in the Fig. 2). This language is exactly $PAL(\{a, b\})$ which demonstrates that recursive Petri net model is a strict extension of the Petri net one.

2.4 Expressive Power of Recursive Petri Nets

Indeed, we show that recursive Petri net languages strictly include the union of Context Free and Petri net languages.

Proposition 1 (Strict Inclusion).
Recursive Petri net languages strictly include the union of Context Free and Petri net languages.

Proof (Sketch). Like the complete proof is not particularly difficult, we only give a sketch of this proof based on simple example.

It is clear that recursive Petri net languages include Petri net language. Then, we begin by the inclusion of Context Free languages. We consider a Context Free language defined by a set of symbols partionned in terminal symbols T and non-terminal symbols N. To each non-terminal symbol $s \in N$ is associated

a non-empty set $\{s_1, s_2, \ldots, s_n\}$ of words on $T \cup N$. A particular non-terminal symbol s_0 is designated as the initial one. A word of the language is obtained by choosing word in the set associated to s_0 and then by iteratively substituting in it a non-terminal symbol by an element of the associated set until the word does not contain any non-terminal symbol.

For a given Context Free language, we can construct a RPN having exactly the same language.

For any set associated to a non-terminal symbol s, we build a particular subnet of the RPN. First, we add a place P_s. Each element of the set associated to s leads to the production of a sequence of transitions. Let s_i be such an element. For each non-terminal symbol r composing s_i, we add an abstract transition labeled by λ and associated to the ordinary marking P_r. For each terminal symbol a, we add an ordinary transition labeled by a. Moreover, to each of these transitions we add an output place and give as input place the output place of the transition associated to the symbol preceding the considered one in the word s_i. Finally, the input place of the first transition is designated to be P_s and a final transition labeled λ is constructed at the output of the last place in the sequence.

The initial extended marking is composed by only one node for which the ordinary marking is P_{s_0} and the language that we consider is $\mathcal{L}(N, tr_0, \{\bot\})$. It can be shown that this language is exactly the considered Context Free language.

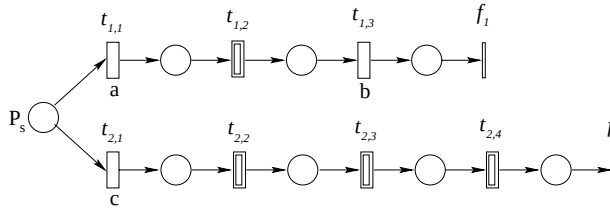


Fig. 3. a RPN modeling a Context Free language

The Fig. 3 illustrates this construction. The part of the RPN corresponding to a non-terminal symbol S associated to the set $\{aSb, cDEF\}$ is given. Symbols in uppercase are considered as non-terminal and the others as terminal. A transition associated to the j^{th} symbol of the i^{th} element of the set is designated by $t_{i,j}$ and the final transition is denoted f_i. As an example, the abstract transition associated to E is denoted $t_{2,3}$. The label λ of the abstract and final transitions has been omitted.

To demonstrate that this inclusion is strict, we present in Fig. 3 (added to the net of the Fig. 1) a RPN for which its language is neither a PN one nor a Context Free one. The ordinary marking associated to the abstract transition t'_0 is the place P of the RPN in the Fig. 1. Moreover, its initial extended marking is composed by a unique node associated to the ordinary marking P_{init}. We consider the language $\mathcal{L}(N, tr_0, \{tr_f\})$ where tr_f is the extended marking composed

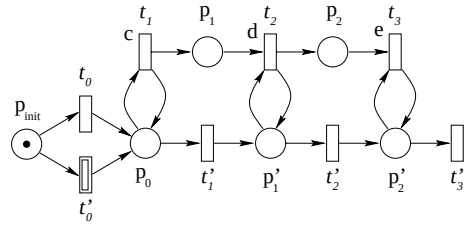


Fig. 4. a particular RPN

by only one node associated to the empty ordinary marking. This language is exactly $\{w_1, w_2\}$ where $w_1 \in Pal(\{a, b\})$ and $w_2 \in \{c^n.d^n.e^n\}$ with $n \geq 0$. This language is not a Petri net one (its projection on $\{a, b\}$ gives $Pal(\{a, b\})$) and is not a Context Free one (its projection on $\{c, d, e\}$ gives $\{c^n.d^n.e^n\}$ with $n \geq 0$) which concludes the proof. □

2.5 Recursive Petri Nets and the Modeling of Faults

In order to give insight to the improvements brought by RPNs, we present in the Fig. 5 an easy way to add faulty behaviors to a Petri net model. The required faulty behavior cleans up the current marking and restarts with the initial marking. To the best of our knowledge, there is no general way to add such a feature to a Petri net. Only ad-hoc mechanisms are proposed for particular cases.

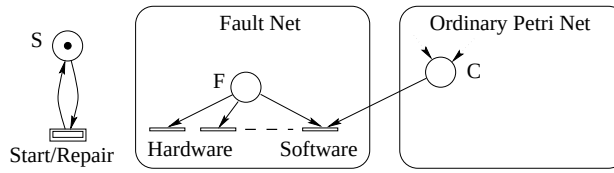


Fig. 5. recursive Petri net modeling faults

Let us look at our figure. The net is divided into three parts. On the right there is the Petri net of the correct behavior (in our formalism, all transitions are elementary). On the center there is the faulty behavior with some spontaneous faults (transition `Hardware`) and some conditioned faults (transition `Software`). For sakeness of the modeling, a control place `F` is added ; all the transitions of this part are final. On the left there is a simple initially marked loop with an abstract transition. The starting marking of this transition is the marking of the Petri net model of the correct behavior plus one token in the control place `F`.

The behavior of the net may be described as follows. Initially and in all the crash states, the extended marking is reduced to one node with the loop place `S` marked. When the abstract transition is fired the correct behavior is "played"

by the new thread. If this thread dies by the firing of a faulty transition, we come back to the initial state and so on.

3 Decidability of the Reachability Problem

The reachability problem consists to determine if a given state is reachable from another one.

This problem has been demonstrated to be decidable for ordinary Petri nets (see [May81,Kos82,Lam88]). The main result of this paper is that the reachability problem can be also decided for recursive Petri nets.

3.1 Branches Structure of a Firing Sequence

First, we characterize the different kinds of behaviors of branches inside a sequence.

Definition 2 (Permanent Branch).
Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'.
A couple of branches $((t_a, tr_a), (t_b, tr_b)) \in Branch(tr) \times Branch(tr')$ denotes a permanent branch *in σ if $v_0(tr_a) = v_0(tr_b)$.*

The previous definition expresses that the node $v_0(tr_a)$ is never removed by the firing of a final transition in $v_0(tr_a)$. Remark that in this case, we have necessary $t_a = t_b$.

If a branch is not permanent and occurs in the final marking then it has been "opened" by an abstract transition.

Definition 3 (Opened Branch).
Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch $(t_b, tr_b) \in Branch(tr')$ denotes an opened branch *in σ if $\forall (t_a, tr_a) \in Branch(tr), v_0(tr_a) \neq v_0(tr_b)$.*

In the same way, if a branch is not permanent and occurs in the initial marking then it has been "closed" by a final transition.

Definition 4 (Closed Branch).
Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch $(t_a, tr_a) \in Branch(tr)$ denotes an closed branch *in σ if $\forall (t_b, tr_b) \in Branch(tr'), v_0(tr_a) \neq v_0(tr_b)$.*

At last, some branches may appear in an intermediate marking and disappear before the final marking.

Definition 5 (Transient Branch).
Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch (t_c, tr_c) of an extended marking tr'' visited by σ is a transient branch *if*
$\forall (t_a, tr_a) \in Branch(tr) \cup Branch(tr'), v_0(tr_a) \neq v_0(tr_c)$.

We want to split the (possibly empty) set of sequences which leads from the initial marking to the final marking depending on the behavior of branches of the initial and final marking. In the decision procedure, we will try to find successively a sequence in each of this subset. Let us note that the number of these subsets is finite. So an admissible combination of branches of the initial marking and final marking binds the permanent branches and the remainding branches are either closed (if they are in the initial marking) or opened (if they are in the final marking).

Definition 6 (Admissible Combination of Branches).
Let tr and tr' be two extended markings, an admissible combination of branches $Cb(tr, tr')$ is defined by $Cb(tr, tr') = \{R_t\}_{t \in T_{ab}}$ such that

- $\forall t \in T_{ab}, R_t \subseteq (branch(tr, t) \cup \perp) \times (branch(tr', t) \cup \perp) \wedge$
- $\forall t \in T_{ab}, \forall b_i \in branch(tr, t), |R_t(b_i, \bullet)| = 1 \wedge$
- $\forall t \in T_{ab}, \forall b_j \in branch(tr', t), |R_t(\bullet, b_j)| = 1 \wedge$
- $\forall t \in T_{ab}, |R_t(\perp, \perp)| = 0$

The meaning of an admissible combination $Cb(tr, tr')$ is the following.

Definition 7 (Combination Respect).
Lets σ be a sequence from tr to tr', then σ respects $Cb(tr, tr')$ iff :
$\forall t \in T_{ab}, \forall b_i \in branch(tr, t), \forall b_j \in branch(tr', t)$ *:*

- $R_t(b_i, \perp)$ *implies σ closes the branch b_i,*
- $R_t(\perp, b_j)$ *implies σ opens the branch b_j and b_j stays opened,*
- $R_t(b_i, b_j)$ *implies b_i stays opened during the firing of σ and corresponds to b_j in tr'.*

The set of possible branch combinations of two extended markings tr and tr' is denoted $CB(tr, tr')$. It is clear that for any sequence there is one and only one admissible combination respected by the sequence.

3.2 Closability of Abstract Transitions

The first step for tackling the reachability problem is to determine whether the tree generated by the firing of an abstract transition may "close" itself.

Definition 8 (Closable Abstract Transition).
An abstract transition t is closable *if there exists a firing sequence from $\Omega(t)$ to $\perp$. Such a sequence is called a* closing sequence *of the abstract transition t.*

From this definition, it is clear that the branches opened in a firing sequence which can be closed in the same sequence, are those which are composed from a closable abstract transition. Moreover, using the sequence depth, we can characterize some minimal closing sequences.

Definition 9 (Minimal Closing Sequence).
A closing sequence σ of an abstract transition t is said minimal *if for all closing sequence σ' of t, we have $Depth(\sigma) \leq Depth(\sigma')$.*

For a given closable abstract transition t, we denote by $Order(t)$ the depth of its minimal closing sequences. The next lemma exhibits a structural property of some minimal closing sequences.

Lemma 10 (Existence of Particular Minimal Closing Sequences).
Let t be an abstract closable transition, there exists a minimal closing sequence σ such that: If σ' denotes the prefix of σ where the last transition (a final one) has been deleted then σ' has no opened branches.

Proof. Let us take a closing sequence. If there is an opened branch before the last firing, then we can delete the firing which opens this branch and all the subsequent firings in this branch. Indeed, the only node modified by the removing is the root node where all its marking after the opening of the branch are now increased with the input tokens of the abstract transition. Thus the new sequence is also firable. The new sequence has at most the same depth as the previous one. Iterating this removing on all the opened branches, we obtain the required minimal closing sequence. □

Lemma 11 (Branches Order).
If there exists an abstract transition $t \in T_{ab}$ such that $Order(t) = n$ (with $n > 1$) then there exists an abstract transition t' such that $Order(t') = n - 1$

Proof. Let σ be a minimal closing sequence of t fulfilling the condition of lemma 10 and $\{tr_1, \ldots, tr_m\}$ the extended markings of depth n reached by σ. Let $\{t_{i,j}\}$ be the transitions labeling the branches of tr_i.

By the definition of σ, we are ensured that $\forall i, j, Order(t_{i,j}) \leq n-1$. Moreover the condition of the lemma 10 ensures that all the branches are closed before the firing of the last transition (the final one) in the root.

Suppose that $\forall i, j, Order(t_{i,j}) < n - 1$, then we can replace the sequence σ by a sequence σ' where the firings in the branch which follow its creation are replaced by some minimal closing sequence of order $< n - 1$. This sequence σ' has an order $< n$ which contradicts the hypothesis of minimality of σ. □

The Algorithm 3.1 computing the set of closable abstract transitions is based on the Lemma 11. From a given recursive Petri net, it returns the corresponding set of closable abstract transitions.

Proposition 12 (Closable Abstract Transitions).
The Algorithm 3.1 computes exactly the set of closable abstract transitions of the net N.

Proof. By definition, an abstract transition of order equal to 1 has an associated minimal closing sequence during which no branch is opened and then no abstract

Algorithm 3.1 Closable

$TransitionSet$ **Closable**($RPN\ N$)
begin
  $Net = N \setminus \{T_{ab} \cup T_{fi}\}$;
  $Computed = \emptyset$;
  $S_{fi} = \{m \in \mathbb{N}^P \mid \exists t \in T_{fi}, m \geq W^-(\bullet, t)\}$;
  $i = 0$;
  do
    $i = i + 1$;
    $New = \emptyset$;
    forall $t \in T_{ab} \setminus Computed$ **do**
      if $ElemDecide(Net, \Omega(t), S_{fi})$ **then**
        $New = New \cup \{t\}$;
        $Order(t) = i$;
      fi
    od
    forall $t \in New$ **do**
      t_{eq} = an elementary transition such that
        $W^-(\bullet, t_{eq}) = W^-(\bullet, t) \wedge W^+(\bullet, t_{eq}) = W^+(\bullet, t)$;
      $Net = Net \cup \{t_{eq}\}$;
    od $Computed = Computed \cup New$;
  while $(New \neq \emptyset)$;
  return $Computed$;
end

transition is fired. Hence, we have to decide if, for a given abstract transition t, a marking from which a final transition is firable can be reached from $\Omega(t)$. The set S_{fi} represents the semi-linear set of markings from which a final transition is firable. The ordinary net Net is obtained removing all abstract and final transitions. A closable abstract transition of order equal to 1 is determined by deciding if there exists a marking of S_{fi} reachable in Net from $\Omega(t)$. In ordinary Petri nets, the reachability of a semi-linear set reduces straightforwardly to the reachability of a finite set of markings. The call $ElemDecide((Net, \Omega(t), S_{fi})$ return $true$ if the ordinary net Net can reach a least a marking of S_{fi} from $\Omega(t)$ and $false$ otherwise.

As seen in the demonstration of the Lemma 11, we know that the closable abstract transitions of order equal to n have an associated minimal closing sequence containing branches of order strictly less than n. Moreover, these branches are transient in the subsequence which precedes the firing of the last transition. Moreover, only the marking of the root is relevant to reach S_{fi}. Then we can mimic the consequence at the root level of behavior of these branches by adding to the net, elementary transitions equivalent to the closable abstract transitions of order strictly less than n. A similar construction is used in Proposition 16 and explained there in more details.

Finally, as the number of abstract transitions is finite and each sucessfull round of the external loop picks at least a new one, the algorithm stops. □

For a given recursive Petri net N, we denote by N_{ord} the ordinary net Net obtained at the termination of the Algorithm 3.1.

3.3 Analysis of Sequences Structure

We enumerate four propositions (one per category of branches) which are useful to reduce the problem of existence of a sequence to the existence of other (and simpler in some sense) sequences.

Proposition 13 (Permanent Branch Condition).
Let tr and tr' be two extended markings. Let $Cb(tr, tr')$ be an admissible combination of branches and t an abstract transition. There exists a sequence σ from tr to tr' which respects $Cb(tr, tr')$ with a permanent branch $R_t(b_i, b_j)$ iff $\exists \sigma', \sigma''$ such that

$$(M(v_0(tr)), Branch(tr) \setminus b_i) \xrightarrow{\sigma'} (M(v_0(tr')), Branch(tr') \setminus b_j) \wedge$$
$$b_i \xrightarrow{\sigma''} b_j$$

Proof. The demonstration is essentially based on the semantics of recursive Petri nets. If the branch b_i is permanent then the sequence σ does not contain a firing of an abstract transition opening the branch and any firing of final transition in the root of the branch. Then, all the firings in the branch b_i are independent from the ones outside of the branch.

We construct a sequence σ' by projecting σ on the firings which do not concern b_i and a sequence σ'' by projecting σ on the firings concerning the branch b_i. Due to the fact that the firings inside b_i are independent to the ones outside (from the semantics of recursive Petri nets), if σ is firable then the sequence $\sigma'.\sigma''$ is also firable and leads from tr to tr'.

Moreover, the sequence $\sigma''.\sigma'$ has the same properties.

Finally, it is clear that the existence of firing sequences σ' and σ'' is a sufficient condition for reaching tr' from tr via $\sigma'.\sigma''$. □

This proposition is illustrated in the Fig. 6.

Proposition 14 (Opened Branch Condition).
Let tr and tr' be two extended markings. Let $Cb(tr, tr')$ be an admissible combination of branches and t an abstract transition. There exists a sequence σ from tr to tr' which respects $Cb(tr, tr')$ with an opened branch $R_t(\perp, b_j)$ iff $\exists \sigma', \sigma''$ such that

$$tr \xrightarrow{\sigma'} (M(v_0(tr')) + W^-(\bullet, t), Branch(tr') \setminus \{b_j\}) \wedge$$
$$\Omega(t) \xrightarrow{\sigma''} b_j$$

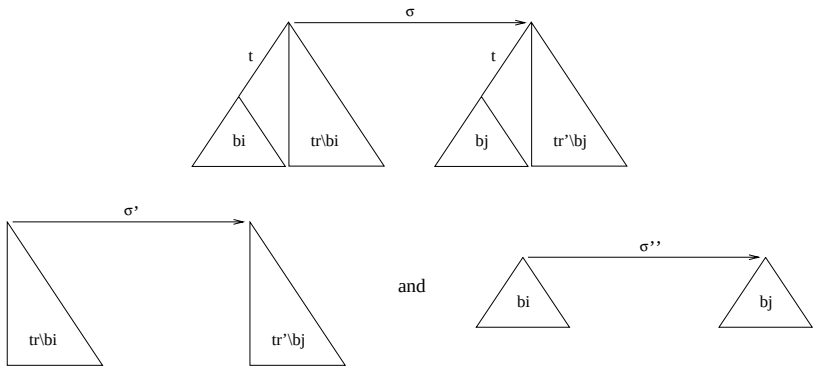


Fig. 6. permanent branch

Proof. t is the abstract transition which opens the branch b_j. Let σ_a be the part of σ preceding the b_j-opening occurrence of t in σ and σ_b the part of σ following this occurrence. We have $\sigma = \sigma_a.t.\sigma_b$.

The branch b_j is a permanent branch of σ_b. Applying the Proposition 13, we construct a sequence σ_{b1} by projecting σ_b on the firings which do not concern the branch b_j and a sequence σ_{b2} by projecting σ_b on the firings concerning b_j.

The sequence $\sigma_a.t.\sigma_{b1}.\sigma_{b2}$ is firable.

Due to the fact that the firing of t only consumes tokens in the root of the extended markings, the sequence $\sigma_a.\sigma_{b1}.t.\sigma_{b2}$ is also firable and leads from tr to tr'.

Moreover, like the firing of t leads to the opening of the branch $(t, \Omega(t))$ and the firings in σ_{b2} only concern b_j, we have $\Omega(t) \xrightarrow{\sigma_{b2}} b_j$.

Since all the firings in σ which do not concern b_j are in $\sigma_a.\sigma_{b1}$, we have $tr^{\sigma_a.\sigma_{b1}}\!\!\longrightarrow (M(v_0(tr')) + W^-(\bullet, t), Branch(tr') \setminus \{b_j\})$.

Finally, it is clear that the existence of firing sequences σ' and σ'' is a sufficient condition for the reachability via $\sigma'.\mathrm{t}.\sigma''$. □

This proposition is illustrated in the Fig. 7.

Proposition 15 (Closed Branch Condition).

Let tr and tr' be two extended markings. Let $Cb(tr, tr')$ be an admissible combination of branches and t an abstract transition. There exists a sequence σ from tr to tr' which respects $Cb(tr, tr')$ with a closed branch $R_t(b_i, \perp)$ iff $\exists \sigma', \sigma''$ such that

$$b_i \xrightarrow{\sigma'} \perp \wedge$$
$$(M(v_0(tr)) + W^+(\bullet, t), Branch(tr) \setminus \{b_i\}) \xrightarrow{\sigma''} tr'$$

Proof. Lets t_{cl} be the final transition closing the branch b_i. Let σ_a be the part of σ preceding the b_i-closing occurrence of t_{cl} in σ and σ_b the part of σ following this occurrence. We have $\sigma = \sigma_a.t_{cl}.\sigma_b$.

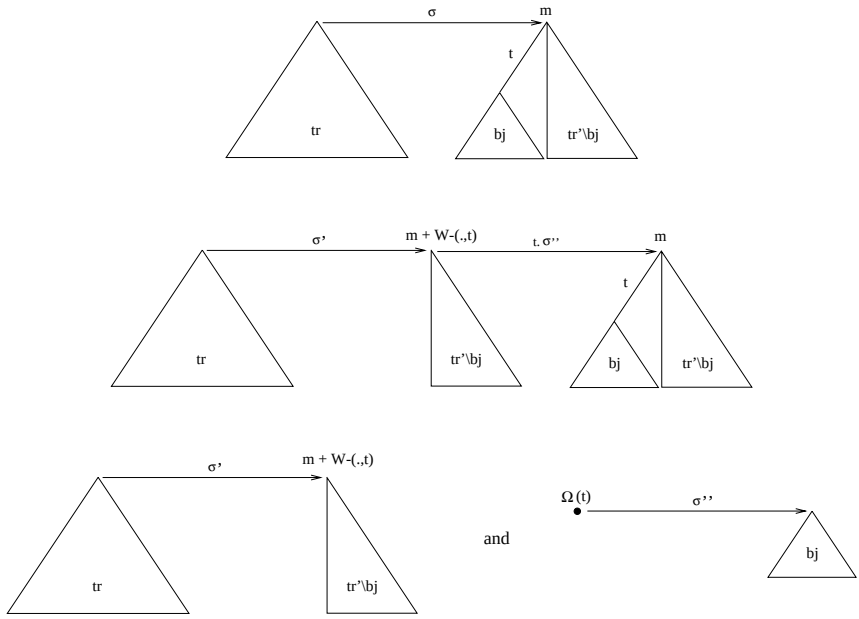


Fig. 7. opened branch

The branch b_i is a permanent branch of σ_a. Applying the Proposition 13, we construct a sequence σ_{a1} by projecting σ_a on the firings which concern the branch b_i and a sequence σ_{a2} by projecting σ_b on the firings not concerning b_i.

The sequence $\sigma_{a1}.\sigma_{a2}.t_{cl}.\sigma_b$ is firable.

Due to the fact that the firing of t_{cl} only produces tokens in the root of the extended markings, the sequence $\sigma_{a1}.t_{cl}.\sigma_{a2}.\sigma_b$ is also firable and leads from tr to tr'.

Because all the firings in σ which do not concern b_i are in $\sigma_{a2}.\sigma_b$, we have $tr^{\sigma_{a1}.t_{cl}} \xrightarrow{} (M(v_0(tr)) + W^+(\bullet, t), Branch(tr) \setminus \{b_i\})$.

Finally, it is clear that the existence of firing sequences σ' and σ'' is a sufficient condition for reachability via $\sigma'.\sigma''$. □

This proposition is illustrated in the Fig. 8.

Proposition 16 (Transient Branch Elimination).
Let tr and tr' be two extended markings composed by only one node. There exists σ a sequence from tr to tr' iff there is a sequence in the ordinary Petri net N_{ord} leading from $M(v_0(tr))$ to $M(v_0(tr'))$.

Proof. Lets t_{cl} be the final transition closing some transient branch b_i of σ opened by an abstract transition t. Let σ_a be the part of σ preceding the firing of t, σ_b the part of σ enclosed by the opening-firing of t of the branch and the closing-firing of t_{cl} and σ_c be the part of σ following the firing of t_{cl}. We have $\sigma = \sigma_a.t.\sigma_b.t_{cl}.\sigma_c$.

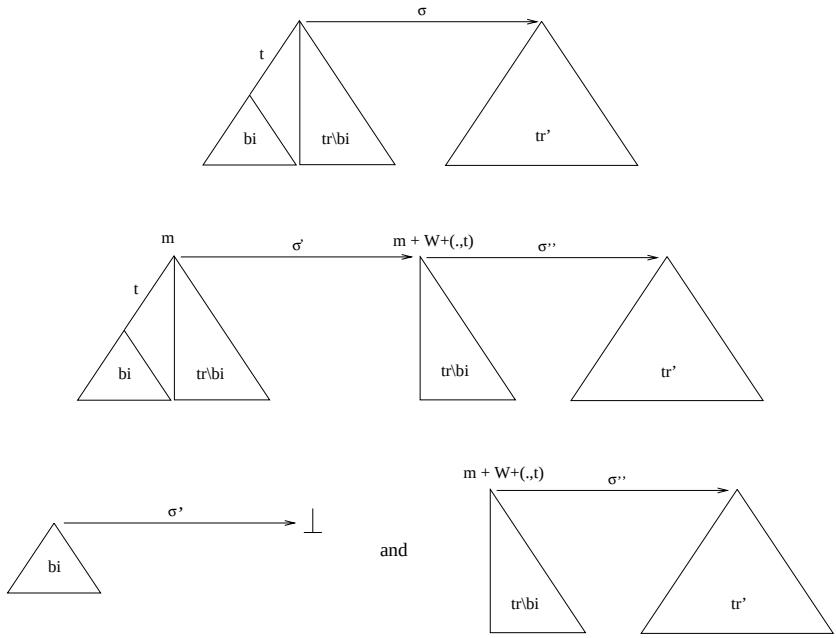


Fig. 8. closed branch

The branch b_i is a permanent branch of σ_b. Applying the Proposition 13, we construct a sequence σ_{b1} by projecting σ_b on the firings which do not concern the branch b_i and a sequence σ_{b2} by projecting σ_b on the firings concerning b_i.

The sequence $\sigma_a.t.\sigma_{b1}.\sigma_{b2}.t_{cl}.\sigma_c$ is firable.

Due to the fact the firing of t only consumes tokens in the root of the extended markings, the sequence $\sigma_a.\sigma_{b1}.t.\sigma_{b2}.t_{cl}.\sigma_c$ is also firable and leads from tr to tr'.

Because the sequence σ_{b2} only concerns the branch b_i opening by t, its firings do not have any effect on the nodes of extended marking reached by $\sigma_a.\sigma_{b1}$. The modifications on these nodes done by the sequence $t.\sigma_{b2}.t_{cl}$ concern only the root node and are the consuming of $W^-(\bullet, t)$ tokens by t and the producing of $W^+(\bullet, t)$ tokens by t_{cl}. So at the root level, the sequence $t.\sigma_{b2}.t_{cl}$ can be simulated by t_{eq} which belongs to N_{ord} as t has a closing sequence. Iterating the substitution for all transient branches gives a sequence in N_{ord}.

Finally, it is clear that a sequence in N_{ord} can be transformed in a sequence for N by substituting the firing of any t_{eq} by the firing of t followed by a closing sequence of t. □

3.4 The Decision Procedure

The Algorithm 3.2 is developed from the previous propositions.

Algorithm 3.2 Decide

Bool **Decide**(N, tr, tr')
begin
  if $(tr == \bot)$ **then**
    return $tr' == \bot$;
  if $(tr' == \bot)$ **then**
    $m = M(v_0(tr))$;
    forall $(t, tr_i) \in Branch(tr)$ **do**
      if $Decide(N, tr_i, \bot)$ **then**
        $m = m + W^+(\bullet, t)$;
      fi
    od
    $S_{fi} = \{m \in \mathbb{N}^P \mid \exists t \in T_{fi}, m \geq W_-(\bullet, t)\}$;
    return $ElemDecide(N_{ord}, m, S_{fi})$;
  fi
  forall $Cb(tr, tr') \in CB(tr, tr')$ **do**
  // with $Cb(tr, tr') = \{R_t\}_{t \in T_{ab}}$
    forall $R_t((t, tr_i), (t, tr_j))$ **do** // permanent branch
      if $\neg Decide(N, tr_i, tr_j)$ **then**
        goto $NextCombination$;
      fi
    od
    forall $R_t(\bot, (t, tr_j))$ **do** // opened branch
      if $\neg Decide(N, \Omega(t), tr_j)$ **then**
        goto $NextCombination$;
      fi
    od
    forall $R_t((t, tr_i), \bot)$ **do** // closed branch
      if $\neg Decide(N, tr_i, \bot)$ **then**
        goto $NextCombination$;
      fi
    od
    $m_1 = M(v_0(tr)) + \sum_{R_t(\bullet, \bot)} W^+(\bullet, t)$;
    $m_2 = m_0(tr') + \sum_{R_t(\bot, \bullet)} W^-(\bullet, t)$;
    if $ElemDecide(N_{ord}, m_1, m_2)$ **then**
      return *true*;
    fi
NextCombination:
  od
  return *false*;
end

Theorem 17 (Reachability Problem).
Let tr and tr' two extended markings of a recursive Petri net $N = \langle P, T, W^-, W^+, \Omega, tr_0 \rangle$. tr' is reachable from tr iff $Decide(N, tr, tr')$ returns true.

Proof. The demonstration is done recursively on $depth(tr) + depth(tr')$.

If $depth(tr) = 0$ then the algorithm returns *true* only if $depth(tr') = 0$ and $false$ otherwise. This statement is insured by the first test of the algorithm.

We make the hypothesis that the algorithm for $depth(tr) + depth(tr') \leq n$ is correct and demonstrate its correctness for $depth(tr) + depth(tr') = n + 1$.

If $depth(tr) \neq 0 \wedge depth(tr') = 0$, we have to decide if the system is able to fire a final transition at the root level. Or equivalently, if the root of tr is able to reach a state from which a final transition is firable. The set S_{fi} represents this set of markings.

If such a sequence exists, we can adapt the Lemma 10 to it and then, there exists a minimal sequence having no opened branch.

However, a branch opened in tr which can be closed from tr, can increase the number of token in the marking associated to $v_0(tr)$ by producing $W^+(\bullet, t)$ tokens (where t is the abstract transition associated to the branch). Due to the recurrence hypothesis, we can determine these particular branches by recursive calls to the procedure.

Because, adding some tokens to a marking can only increase the number of sequences fired from it, we can arbitrary close these particular branches. The marking m corresponds to this statement. It is clear that the branches opened in tr which can not be closed have no effect on the marking associated to $v_0(tr)$ and then can be ignored.

Finally, because the searched sequence can perform some transient branches and applying the Proposition 16, decide if the system can reach a marking of S_{fi} from m is equivalent to decide if the ordinary net N_{ord} is able to reach a marking of S_{fi} from m. The demonstration of this point is related to the Proposition 16 and the reachability problem for ordinary nets is known to be decidable ([May81,Kos82,Lam88]).

If $depth(tr) \neq 0 \wedge depth(tr') \neq 0$ and because the number of combinations is finite, decide if there exists a firing sequence in N leading from tr to tr' is equivalent to decide if there exist a firing sequence σ in N leading from tr to tr' and there exists an admissible combination $Cb(tr, tr')$ such that σ respects $Cb(tr, tr')$.

The main loop of the algorithm corresponds to this statement.

Then, to decide if there exists a sequence in N is equivalent to decide if there exists one showing some permanent, opened, closed and transient branches. The considered admissible combination Cb determines the permanent, opened and closed branches.

The three internal loops correspond to the treatment of these kinds of branch.

The demonstration of there correctness is directly related to the Propositions 13, 14 and 15 and to the induction hypothesis.

When all the permanent, opened and closed branches have been treated, the decision of the reachability can be restricted to a decision in an ordinary net. If

we consider that all the closed branches are effectivly closed, we have to reach the state from which all the opened branches can be opened. The markings m_1 and m_2 correspond to these two particular points of the sequence. Because, the sequence can perform some transient branches, the decision must be done for the ordinary net N_{ord} (Proposition 16). □

4 Conclusion

In this work, we have studied theoretical features of recursive Petri nets. We have first shown that they are a strict extension of the model of Petri nets as they are able to recognize the palindrom language and that the languages of RPNs strictly include the union of Petri net and Context Free languages. Moroveover, we have illustrated their modelling capability by giving a simple method to model faults in a system whereas a similar modelling is not known to be possible for ordinary Petri nets.

Then, we have proven that the reachability is decidable for RPNs and we have given an algorithm which reduces the problem to some (quite numerous !) applications of the decision procedure for the ordinary Petri nets.

We plan to extend our studies in two different ways. On the one hand we want to add new features for recursive Petri nets and examine whether the reachability problem remains decidable. We are mainly interested to introduce some context when a thread is initiated (e.g. the starting marking could depend from the depth in the tree). On the other hand, we would study some more complex properties (like home state or properties specified by a temporal formula). Since the redaction of this paper, new results on RPNs have been stated. In particular, we have proven that RPN languages are recursive [HP99b] and that any Turing machine can be simulated by synchronizing a RPN with a finite automaton [HP99a]. This last result has multiple consequences. One of them being that the emptiness of the intersection of a regular language and a RPN language is undecidable. Hence, it leaves little hope to general model checking as done in [Esp97] for Petri nets. However, this negative result does not preclude checking of particular properties (such as special kinds of fairness [Yen92]) and we are working in such a direction.

At last, it would be interesting to combine this model with usual characteristics (such like the colours) in order to increase its application area.

References

DFS98. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. *(ICALP'98) L.N.C.S*, 1443:103–115, July 1998.

Esp97. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

HP99a. S. Haddad and D. Poitrenaud. On the expressive power of recursive Petri net. Submitted to the 12th International Symposium on Fundamentals of Computation Theory, FCT'99, August 1999.

HP99b. S. Haddad and D. Poitrenaud. Recursive Petri net languages are recursive. Submitted to the 10th International Conference on Concurrency Theory, CONCUR'99, August 1999.

Jan79. M. Jantzen. On the hierarchy of Petri net languages. *RAIRO*, 13(1):19–30, 1979.

Kos82. S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th Annual Symposium on Theory of Computing*, pages 267–281, 1982.

Lam88. J. L. Lambert. Some consequences of the decidability of the reachability problem for Petri nets. In *Advances in Petri Nets 1988*, volume 340 of *LNCS*, pages 266–282, Berlin, Germany, June 1988. Springer Verlag.

May81. E.W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. 13th Annual Symposium on Theory of Computing*, pages 238–246, 1981.

Rei95. K. Reinhardt. Reachability in Petri nets with inhibitor arcs. Unpublished manuscript. See www-fs.informatik.uni-tuebingen.de/ reinhard, 1995.

SH95. A. El Fallah Seghrouchni and S. Haddad. A formal model for coordinating plans in multiagents systems. In *Proceedings of Intelligent Agents Workshop*, Oxford United Kingdom, November 1995. Augusta Technology Ltd, Brooks University.

SH96. A. El Fallah Seghrouchni and S. Haddad. A recursive model for distributed planning. In *Second International Conference on Multi-Agent Systems*, Kyoto, Japon, December 1996.

Val78a. R. Valk. On the computational power of extended Petri nets. *(MFCS'78) L.N.C.S*, 64:526–535, September 1978.

Val78b. R. Valk. Self-modifying nets, a natural extension of Petri nets. *(ICALP'78) L.N.C.S*, 62:464–476, July 1978.

Yen92. H-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96:119–137, 1992.

Petri Net Theory - Problems Solved by Commutative Algebra

Christoph Schneider[1] and Joachim Wehler[2]

[1]cke-schneider.de data service, Paul Robeson Straße 40, D-10439 Berlin, Germany
info@cke-schneider.de
[2]Döllingerstraße 35, D-80639 München, Germany
wej@softlab.de

Abstract. The paper deals with the computation of flows in coloured nets and with the potential reachability of markings over the integers in p/t nets. We introduce Artin nets as a subclass of coloured nets, which can be handled by methods from Commutative Algebra. As a first result we develop an algorithm for the explicit computation of flows in Artin nets, which is supported by existing tools. Concerning reachability in p/t nets we prove a refined rank condition as a second result.

Keywords. Coloured Petri net, Artin net, commutative net, flow, reachability, Gröbner theory.

1 Introduction

The present work deals with the following problems from Petri net theory:

Computation of flows in coloured nets.

Potential reachability resp. nonreachability of markings over $\boldsymbol{Z}$ in p/t systems.

Our results are:

For a subclass of coloured nets (Artin nets) we develop an algorithm, which computes generators of the module of flows (Algorithm 5.2). This algorithm is supported by existing tools (Remark 5.3). Our results generalize in several aspects the results of *Couvreur* and *Martinez* about commutative nets ([CM1990]): The colour functions are not restricted to diagonalizable matrices (Definition 3.1) and we show, how to obtain a minimal set of generators of flows (Example 5.5).

For p/t systems we prove a refined rank condition for the incidence map, which characterizes potential reachability over $\boldsymbol{Z}$ (Theorem 6.4). This results provides a second criterion, complementary to the characterization by modulo-invariants given by *Desel*, *Neuendorf* and *Radola* ([DNR1996]). The present formulation can serve as starting point for generalizing the criterion to Artin nets and other commutative nets.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 248-267, 1999.
© Springer-Verlag Berlin Heidelberg 1999

We apply the following methods from Commutative Algebra:

Gröbner theory for rings of polynomials. The present paper uses this method for the first time to solve Petri net problems.

Theory of modules over principal ideal domains. This theory has been applied to Petri nets already by Desel, Neuendorf and Radola.

Both methods transcend Linear Algebra for vector spaces considerably. Although we often use terms, which probably are unfamiliar in the context of Petri nets, you will get concrete numerical results by the above methods and the supporting tools. Some proofs or remarks require non-standard prerequisites. They are typed in smaller fonts.

Obviously every coloured net can be unfolded into an equivalent p/t net. But in our opinion the descent to low level nets can not qualify as the method of choice to answer high level questions. Therefore one has to leave the theory of vector spaces and has to apply more sophisticated methods from mathematics. We will use the concept of *coloured nets* from [Jen1992].

2 Linear Problems in Coloured Homogeneous Nets

Linear Algebra and Linear Programming theory are the mathematics of p/t nets. Matrix theory solves systems of linear equations over a field and determines e.g. the flows of p/t nets. Coloured nets are controlled by ground rings, which are more general than the field of rationals $\boldsymbol{Q}$ or the ring of integers $\boldsymbol{Z}$. Their rings are not necessarily commutative and the colour modules can vary from place to place and from transition to transition.

Currently there is no universal mathematical method, capable to analyze all kinds of coloured nets. The scope of each particular method restricts to subclasses. In the present chapter we choose the subclass of homogeneous nets as a suitable frame for fixing our algebraic notations. This class has been introduced by Couvreur and Martinez ([CM1990]).

Petri net theory is intimately linked with non-negative coefficients. But in the present paper we will restrict to coarser structures with coefficients from a ring R: We do not consider bags but R-modules, replacing linear algebra over a field by the theory of modules over a ring.

2.1 Remark (Notations from Algebra). Denote by R a ring.

i) For an arbitrary set X we will denote by X_R the *free R-module with base X*

$$X_R := \{ \sum_{x \in X} n_x x : n_x \in R \text{ and } n_x = 0 \text{ for at most finitely many } x \} .$$

Elements of X_R are all finite weighted sums of elements from X with weights from R.

ii) For two R-modules V, W we denote by

$$\mathrm{Hom}_R(V, W) := \{ f: V \to W : f \text{ is R-linear} \}$$

the set of *morphisms* between V and W. If V = W then we use the notation

$$\mathrm{End}_R(V) := \mathrm{Hom}_R(V, V).$$

The R-module $\mathrm{Hom}_R(V, R) =: V^*$ is called the *dual* of V.

2.2 Definition (Homogeneous Net). A coloured net $N = (T, P, C, w^+, w^-)$ with transitions T, places P, pre- resp. post-conditions w^+ resp. w^- is called *homogeneous*, iff all places and transitions share a common colour set C.

We stress, that we do not require the same token flow for all bindings of a given transition. Plain p/t nets are the special class of homogeneous nets with a single colour, i.e. C = { }. Every homogeneous net shows up with an intrinsic algebra, much more specific than the ring of definition **Z**: Our approach studies linear questions therefore over the *colour algebra* of the net.

2.3 Definition. (Colour Algebra of a Homogeneous Net) The *incidence map* of a homogeneous net $N = (T, P, C, w^+, w^-)$ is the **Z**-bilinear map

$$w := w^+ - w^- : T_{\mathbf{Z}} \times P_{\mathbf{Z}} \quad \mathrm{End}_{\mathbf{Z}}(C_{\mathbf{Z}}).$$

The local images

$$w^+(t, p), w^-(t, p) \quad \mathrm{End}_{\mathbf{Z}}(C_{\mathbf{Z}}), (t, p) \quad T \times P,$$

are called the *colour functions* of N; they generate the *colour algebra*

$$A_{\mathbf{Z}} := \mathbf{Z} [w^+(t, p)_{(t,p) \ TxP}, w^-(t, p)_{(t,p) \ TxP}],$$

which is an associative, but not necessarily commutative subalgebra of $\mathrm{End}_{\mathbf{Z}}(C_{\mathbf{Z}})$.

The following remark fixes our notation concerning the morphisms, which originate from the incidence map.

2.4 Remark (Incidence Map of a Homogeneous Net). Denote by $N = (T, P, C, w^+, w^-)$ a homogeneous net.

i) We introduce

$C_1(N) := P_{\mathbf{Z}}$ the **Z**-module of places and $C_0(N) := T_{\mathbf{Z}}$ the **Z**-module of transitions.

For an arbitrary **Z**-module V we define the **Z**-modules

$C_i(N, V) := C_i(N) \ _{\mathbf{Z}} V$, and $C^i(N, V) = \mathrm{Hom}_{\mathbf{Z}}(C_i(N), V)$, i = 0, 1.

The **Z**-modules

$$C_0(N, C_{\mathbf{Z}}) \text{ resp. } C^1(N, C_{\mathbf{Z}})$$

are called the module of *steps with integer coefficients* resp. the module *of markings with integer coefficients*. The general element

$$= \ _{t \ T} \ _t t \quad b_t \quad C_0(N, C_{\mathbf{Z}})$$

is a linear combination of transitions t T and corresponding bindings b_t C with integer coefficients $_t$ **Z**. Similarly the general element

$$= \ _{p \ P} \ _p p^* \quad c_p \quad C^1(N, C_{\mathbf{Z}})$$

is a linear combination of the dual functionals p^* $\mathrm{Hom}_{\mathbf{Z}}(P_{\mathbf{Z}}, Z)$ and corresponding colours c_p C with integer coefficients $_p$ **Z**.

ii) The incidence map is equivalent to each of the following two morphisms:

$$w_P : C_1(N) \quad C^0(N, A_{\mathbf{Z}}) \quad C^0(N, \mathbf{Z}) \ _{\mathbf{Z}} A_{\mathbf{Z}}, \ w_P(p) := \ _{t \ T} t^* \quad w(t, p)$$

w_T: $C_0(N)$ $C^1(N, A_Z)$ $C^1(N, Z)$ $_Z A_Z$, $w_T(t) :=$ $_{p\ P} p^*$ $w(t, p)$.

iii) By forming the tensor product with A_Z we extend also the domains of these maps to A_Z-modules. We obtain two morphisms of A_Z-modules

w_P: $C_1(N, A_Z)$ $C^0(N, A_Z)$, $w_P(p$ $f) :=$ $_{t\ T}$ (t^* $fow(p, t)$), p P, f A_Z

w_T: $C_0(N, A_Z)$ $C^1(N, A_Z)$, $w_T(t$ $f) :=$ $_{p\ P}$ (p^* $fow(p, t)$), t T, f A_Z.

Obviously we can replace in both maps the subalgebra A_Z by the full algebra of endomorphisms $End_Z(C_Z)$ and obtain morphisms between $End_Z(C_Z)$-modules. For these and other morphisms derived from the total incidence map we will always use the same notation w_P resp. w_T.

The following Lemma 2.5 prepares the proof of Proposition 2.8.

2.5 Lemma (Adjointness of the Incidence Maps). Denote by $N = (T, P, C, w^+, w^-)$ a homogeneous net.

i) The canonical evaluation

$$End_Z(C_Z) \times C_Z \quad C_Z, (f, c) \mapsto f(c)$$

induces two Z-bilinear maps, which are defined on generators as

$< , >_P$: $C_1(N, End_Z(C_Z)) \times C^1(N, C_Z)$ C_Z, $< p$ $f,$ $>_P := f($ $(p))$

resp. $< , >_T$: $C^0(N, End_Z(C_Z)) \times C_0(N, C_Z)$ C_Z, $<$ $, t$ $b >_T :=$ (t) (b).

ii) With respect to these bilinear maps the incidence maps

w_P: $C_1(N, End_Z(C_Z))$ $C^0(N, End_Z(C_Z))$, $w_P(p$ $f) :=$ $_{t\ T}$ (t^* $fow(p, t)$)

w_T: $C_0(N, C_Z)$ $C^1(N, C_Z)$, $w_T(t$ $b) :=$ $_{p\ P}$ p^* $w(t, p)$ (b)

are *adjoint*, i.e. for all x $C_1(N, End_Z(C_Z))$ and y $C_0(N, C_Z)$ holds

$$< x, w_T(y) >_P = < w_P(x), y >_T \quad C_Z.$$

A necessary criterion for an arbitrary marking m_{post} to be reachable in the Petri net (N, m_{pre}) is the fact, that both markings satisfy the marking equation. Due to this linear inhomogeneous equation Petri net theory is the refinement of a linear theory. In general the refinement is strict: The fine structure of a Petri net is determined by a condition on positivity, which is required by the firing rule.

2.6 Remark (Marking Equation). If the binding element (t, b) $T \times C$ of a homogeneous net $N = (T, P, C, w^+, w^-)$ occurs at the marking

$$m_{pre} \quad C^1(N, C_Z),$$

then it creates the new marking m_{post} according to the *marking equation*

$$m_{post} = m_{pre} + w_T(t \quad b) \quad C^1(N, C_Z).$$

The statement generalizes from (t, b) to arbitrary Parikh vectors from $C_0(N, C_Z)$.

For coloured nets one usually considers flows with coefficients from the whole ring of endomorphisms. For the subclass of homogeneous nets it is helpful to focus on flows with coefficients from the colour algebra first, cf. Remark 3.7.

2.7 Definition (Flows of a Homogeneous Net). For a homogeneous net $N = (T, P, C, w^+, w^-)$ with colour algebra A_Z we define the

A_Z-module of *P-flows*[1] of N with coefficients from A_Z as

$$Z_1(N, A_Z) := \ker [w_P: C_1(N, A_Z) \quad C^o(N, A_Z)]$$

and the A_Z-module of *T-flows* of N with coefficients from A_Z as

$$Z_0(N, A_Z) := \ker [w_T: C_0(N, A_Z) \quad C^1(N, A_Z)].$$

An analogous definition is obtained by replacing A_Z by $End_Z(C_Z)$ or by the rational colour algebra

$$A_Q := A_Z \quad_Z Q.$$

P-flows induce a conserved sum of colours as follows at once from the adjointness of the two incidence maps.

2.8 Proposition (P-Flows and Conservation Law). Let $N = (T, P, C, w^+, w^-)$ be a homogeneous net and consider a P-flow

$$Z_1(N, A_Z)$$

with coefficients from the colour algebra A_Z. Then for any two markings

$$m_{pre}, m_{post} \text{ with } m_{post} \quad [m_{pre}>$$

holds the *conservation law*

$$< \ , m_{post} >_P = < \ , m_{pre} >_P \quad C_Z.$$

Proof. We assume without loss of generality that the new marking m_{post} is created by the occurrence of a single binding (t, b) T x C at the marking m_{pre}. According to Lemma 2.5 and because $w_P( ) = 0$ we have

$$< \ , m_{post} >_P = < \ , m_{pre} + w_T(t \quad b) >_P = < \ , m_{pre} >_P + < \ , w_T(t \quad b) >_P =$$
$$< \ , m_{pre} >_P + < w_P( ), t \quad b >_T = < \ , m_{pre} >_P, \text{ QED.}$$

3 Artin Nets

Artin nets make a first step from p/t nets to a non-trivial class of coloured nets, which can be accessed by mathematical methods. We generalize the concept of commutative nets of Couvreur and Martinez ([CM1990]). Then we define Artin nets

[1] Some authors prefer the name P-invariant, while others like [Jen1992] introduce the notation P-flow and distinguish between flows and invariants. We follow the convention of [Jen1992].

as commutative nets with rational coefficients. In addition to a series of example from [CM1990] also the coloured net of the „dining philosophers“ is an Artin net.

3.1 Definition (Commutative Net, Artin Net). A homogeneous net $N = (T, P, C, w^+, w^-)$ is called *commutative*, iff its colour algebra

$$A_{\boldsymbol{Z}} := \boldsymbol{Z} [w^+(t, p)_{(t,p)\ TxP}, w^-(t, p)_{(t,p)\ TxP}]$$

is commutative. For a commutative net we extend the ring $\boldsymbol{Z}$ to its field of quotients $\boldsymbol{Q}$ and obtain the concept of an *Artin net* with rational colour algebra

$$A_{\boldsymbol{Q}} := A_{\boldsymbol{Z}} \quad {}_{\boldsymbol{Z}}\boldsymbol{Q} = \boldsymbol{Q} [w^+(t, p)_{(t,p)\ TxP}, w^-(t, p)_{(t,p)\ TxP}].$$

We do not require the colour functions $w^+(t, p)$, $w^-(t, p)$ to be diagonalizable. Commutative nets in the sense of [CM1990] are also commutative in the sense of Definition 3.1. They have the additional property, that their colour algebra is reduced.

Throughout the whole paper we illustrate our definitions and results about Artin nets by a fixed example from [CM1990]:

3.2 Example (Commutative Net of Clock Synchronization). Fig.1 shows the net from [CM1990], Section 5.2, which models the synchronization of clocks connected to a virtual ring. The commutative net $N = (T, P, C, w^+, w^-)$ has

transitions $T = \{ t_1, t_2 \}$, places $P = \{ p_1,..,p_4 \}$,

colour set $C = C_1 \times C_2$, $C_i := \boldsymbol{Z}/n_i\boldsymbol{Z}$, $n_i \quad \boldsymbol{N}^*$, $i = 1, 2$,

colour algebra $A_{\boldsymbol{Z}} = \boldsymbol{Z} [sh_1, sh_2]$ with the endomorphisms

$$sh_1 = {}_1 \quad 1 \text{ resp. } sh_2 = 1 \quad {}_2 \quad \mathrm{End}_{\boldsymbol{Z}}(C_{\boldsymbol{Z}}),\ C_{\boldsymbol{Z}} = C_{1,\boldsymbol{Z}} \quad {}_{\boldsymbol{Z}}\, C_{2,\boldsymbol{Z}},$$

induced from the shift maps of the colour components

$$_i: \boldsymbol{Z}/n_i\boldsymbol{Z} \quad \boldsymbol{Z}/n_i\boldsymbol{Z},\ x \mapsto x + 1,\ i = 1,2,$$

and incidence matrix

$$M(w_P) = \begin{matrix} sh_2 & 1 & 1 & sh_2 & 1 \\ 1 & 1 & sh_1 & 0 & \end{matrix} \div \quad M(2 \times 4, A_{\boldsymbol{Z}}).$$

The colours from C_1 represent the stations, while the elements from C_2 are considered as clocks.

One of the main results of this paper is an algorithm, which computes the kernel of the incidence map of an Artin net. We want to stress, that the matrix $M(w_p)$ is not of the usual type with entries from $\boldsymbol{Q}$ or $\boldsymbol{Z}$. Note, that each entry is an endomorphism itself. On a more general level we are faced with the following situation:

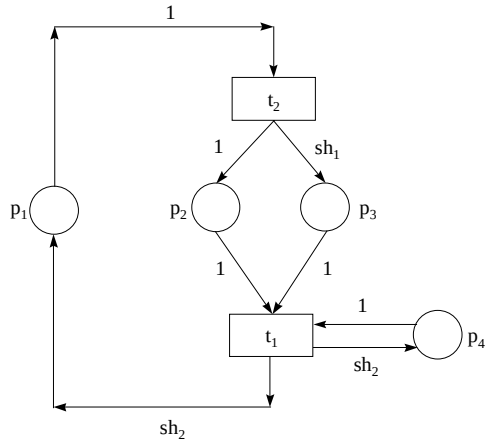


Fig.1. Commutative net of clock synchronization

3.3 Remark (Kernel of a Morphism between A-Modules). Consider a field K and a K-algebra A like the colour algebra of an Artin net. *How to compute the kernel of a morphism f: V W between two A-modules V and W?* We proceed in two steps:

We lift the problem from the ring A to a ring of polynomials.

We apply Gröbner theory over the ring of polynomials.

The second step will be treated in Chapter 4. The first is achieved by the minimal polynomials of the colour functions.

3.4 Remark (From Endomorphisms to Polynomials). i) Every endomorphism f $\mathrm{End}_{\boldsymbol{Q}}(C_{\boldsymbol{Q}})$ of a finite dimensional $\boldsymbol{Q}$-vector space $C_{\boldsymbol{Q}}$ has a unique *minimal polynomial* P_f $\boldsymbol{Q}[\,t\,]$. It is defined as the monic polynomial, such that

$$\boldsymbol{Q}[\,t\,] / < P_f > \quad \boldsymbol{Q}[\,f\,],\ [t] \mapsto f,$$

i.e. such that the rational matrix algebra $\boldsymbol{Q}[\,f\,]$, which is generated by f, equals the quotient of the ring $\boldsymbol{Q}[\,t\,]$ of polynomials in one indeterminate divided by the ideal $< P_f >$, which is generated by P_f.

ii) Analogously the rational matrix algebra $A_{\boldsymbol{Q}}$, which is the adjunction of finitely many colour functions, has the form

$$A_{\boldsymbol{Q}} \quad \boldsymbol{Q}[\,t_1,...,t_m\,] / I,$$

i.e. $A_{\boldsymbol{Q}}$ equals the quotient of a ring $\boldsymbol{Q}[\,t_1,...,t_m\,]$ of polynomials divided by a suitable ideal I $\boldsymbol{Q}[\,t_1,...,t_m\,]$ of polynomials.

iii) The rational colour algebra $A_{\boldsymbol{Q}}$ of an Artin net is a rational *Artin algebra*, i.e. it has finite dimension as vector space over $\boldsymbol{Q}$.

3.5 Example (Colour Algebra of the Clock Synchronization). We continue with the Artin net N from Example 3.2. It has two non-trivial colour functions $sh_j \in End_{\boldsymbol{Z}}(C_{\boldsymbol{Z}})$. Over $\boldsymbol{Q}$ their minimal polynomials are

$$P_j(t) = t^{n_j} - 1 \in \boldsymbol{Q}[t], j = 1,2.$$

The rational colour algebra has the form

$$A_{\boldsymbol{Q}} = \boldsymbol{Q}[sh_1, sh_2] \cong \boldsymbol{Q}[t_1, t_2] / < t_1^{n_1} - 1,\ t_2^{n_2} - 1 >.$$

It is *reduced*, i.e. there are no non-zero elements $f \in A_{\boldsymbol{Q}}$ with $f^n = 0$ for suitable $n \in \boldsymbol{N}$. Each minimal polynomial factors as

$$P_j(t) = (t - 1) \sum_{i=0}^{n_j - 1} t^i .$$

The annihilator ideal of $(sh_j - 1)$ in $\boldsymbol{Q}[sh_1, sh_2]$ is generated by

$$S_j := (1/n_j) \sum_{i=0}^{n_j - 1} sh_j^{\,i} \in A_{\boldsymbol{Q}}, \text{ i.e.}$$

$$\operatorname{ann}(sh_j - 1) := < x \in A_{\boldsymbol{Q}} : x(sh_j - 1) = 0 > = < S_j >, j = 1,2.$$

One defines the n-th cyclotomic polynomial

$$\Phi_n(t) := \Pi_{0 \le k < n} (t - \zeta^k),\ n \in \boldsymbol{N} \text{ and } \zeta \in \boldsymbol{C} \text{ a primitive n-th root of unity.}$$

All cyclotomic polynomials are defined over $\boldsymbol{Z}$ and are irreducible over $\boldsymbol{Q}$. The factorization

$$t^n - 1 = \Pi_{d|n} \Phi_d(t)$$

is a complete split of the left side into a product of irreducible polynomials.

This factorization on the level of polynomials induces a corresponding factorization of the colour algebra. Artin factorization over a field is the key method to reduce linear questions about Artin nets to analogous questions about p/t nets. For an application we refer to Example 5.5.

3.6 Lemma (Artin Factorization). Denote by K a field. Every Artin K-algebra R factors into a finite product

$$R = \Pi_{i=1,\ldots,k} R_i$$

of local Artin K-Algebras R_i (*Artin factors*), i = 1,...,k. For reduced R every factor

$$R_i,\ i = 1,\ldots,k$$

is an extension field of finite degree over the ground field K.

For the concept of a local ring and for the proof cf. [AM1969], Theorem 8.7 resp. [Vas1998], Theorem A.1.4.

For a reduced colour algebra the module of flows with coefficients from the full ring of endomorphisms is already determined by the flows with coefficients from the colour algebra.

3.7 Remark (Ring Extension). Denote by N = (T, P, C, w^+, w) a commutative net. If the rational colour algebra A_Q is reduced, then the natural map

$$Z_i(N, A_Q) \quad {}_{A_Q} \quad End_Q(C_Q) \quad Z_i(N, End_Q(C_Q)),\ i = 1,2,$$

is an isomorphism, i.e. we obtain generators of the flows $Z_i(N, End_Q(C_Q))$ from the generators of $Z_i(N, A_Q)$ by tensoring with endomorphisms.

Proof. By Lemma 3.6 the reduced colour algebra A_Q is a product of fields, hence semisimple. Every module over a semisimple commutative ring is flat, cf. [Bou1972], §2, n°. 4. Hence the torsion product vanishes

$$Tor_1^{A_Q}(-, End_Q(C_Q)) = 0, \text{ QED.}$$

4 Gröbner Theory

The rational colour algebra of an Artin net is a quotient of a ring R of polynomials with rational coefficients. Explicit calculations with free R-modules of finite rank are achieved by Gröbner bases. They reduce statements about polynomials to explicit calculations with monomials. The present chapter gives a short introduction to commutative Gröbner theory. The results will be applied to Petri net theory in Chapter 5. For the sake of simplicity we give the following results only for the ring R = $\boldsymbol{Q}$ [X, Y] of polynomials with 2 indeterminates. But all results hold literally for an arbitrary finite number of indeterminates.

4.1 Definition (Polynomials and Their Leading Terms). i) On the set of terms of R we choose the *homogeneous-lexicographic* order: Two terms

$$T_1 = X^{n_1} Y^{m_1} \text{ and } T_2 = X^{n_2} Y^{m_2} \quad R \text{ satisfy } T_1 > T_2$$

iff either deg $T_1 := n_1 + m_1 >$ deg T_2 or deg T_1 = deg T_2 and $(n_1, m_1) > (n_2, m_2)$ with respect to the lexicographic order of N^2.

ii) With respect to the homogeneous-lexicographic order all terms of a given polynomial

$$f = \sum_{(n,m)\ N^2} c_{n,m} X^n Y^m \quad R,\ c_{n,m} \quad \boldsymbol{Q},$$

can be arranged uniquely in decreasing order. For f 0 the highest term is called the *leading term* LT(f), its coefficient the *leading coefficient* LC(f) of f.

iii) The *S-polynomial* of two elements 0 f, g R is defined as the polynomial

$$S(f, g) := LC(g)\frac{lcm(LT(f), LT(g))}{LT(f)} f \quad LC(f)\frac{lcm(LT(f), LT(g))}{LT(g)} g \quad R$$

with lcm(LT(f), LT(g)) R the least common multiple of the leading terms of f and g.

As a generalization of the Euclidean algorithm we have for multivariate polynomials:

4.2 Lemma (Reduction of a Polynomial). For a given polynomial f and a given set $G = \{ g_1,...,g_n \}$ of polynomials there exists a representation

$$f = \sum_{i=1,...,n} f_i g_i + r$$

with polynomials $f_1,...,f_n$, r R, such that

$LT(f)$ $\max \{ LT(f_i g_i): i = 1,...,n \}$

and no term of r belongs to the ideal $< LT(g_1),...,LT(g_n) >$.

The polynomial r R, the *reduction of f modulo G*, can be computed with a division algorithm.

4.3 Definition (Gröbner Base). A subset $G = \{ g_1,...,g_n \}$ of an ideal I R is called a *Gröbner base* of I, iff the leading terms $LT(g_1),...,LT(g_n)$ generate the *leading ideal* of I

$$LT(I) := < LT(f): f \in I \setminus \{ 0 \} >,$$

which is generated by the leading terms of all non-zero elements from I.

4.4 Proposition (Gröbner Base). i) A Gröbner base of an ideal generates the ideal.

ii) (Buchberger) For a system of generators G of an ideal I R holds the equivalence:

G is a Gröbner base of I.

The reduction modulo G of all S-polynomials of elements from G vanish.

Every element from R has a unique reduction modulo G.

The standard algorithm for the computation of Gröbner bases is due to Buchberger.

4.5 Buchberger Algorithm

Input: Ideal $I = < f_1,...,f_n >$ R. **Output:** Gröbner base $G = \{ g_1,...,g_m \}$ of I.

$G = \{ f_1,...,f_n \}$	
$B = \{ \{f_i, f_j\}: i < j \}$ as a set of unordered pairs	
While B	
	Choose {f, g} B
	B = B \ {f, g}
	Determine a reduction r R of S(f, g) modulo G
	If r 0 then set B = B { {r, h}: h G } and G = G { r }
G is a Gröbner base of I.	

Table 1. Buchberger algorithm

4.6 Example (Gröbner Base). Following Algorithm 4.5 one computes for the ideal

$$I = < X^2, XY + Y^2 > \quad R$$

the Gröbner base

$$G = \{ X^2, XY + Y^2, Y^3 \}.$$

In order to compute the kernel of a linear map we will choose suitable generators of its image and calculate the „syzygies“ of these generators. If the generators form a Gröbner base, then the syzygies can be read off from their S-polynomials. This is the great advantage of Gröbner bases in the present context.

4.7 Definition (Syzygies). A *syzygy* of a finite family $F = \{ f_1,...,f_n \}$ of elements from R is a tuple

$$= (a_1,...,a_n) \text{ of elements from R with } \sum_{i=1,...,n} a_i f_i = 0$$

The syzygies of $F = \{ f_1,...,f_n \}$ form the kernel of the R-linear map

$$f: R^n \quad R \text{ with matrix } M(f) = (f_1,...,f_n).$$

Therefore ker f is called the *module of syzygies* of F.

4.8 Lemma. (Syzygies of a Gröbner Base, Schreyer). Denote by $G = \{ g_1,...,g_n \}$ a Gröbner base of the ideal I R with S-polynomials

$$S(g_i, g_j) = a_{ji} g_i - a_{ij} g_j = \sum_{k=1,...,n} h^{ij}_k g_k, \quad 1 \quad i < j \quad n, \; a_{ij}, h^{ij}_k \quad R.$$

The R-module of syzygies of G is generated by the elements

$$_{ij} := a_{ji} e_i - a_{ij} e_j - \sum_{k=1,...,n} h^{ij}_k e_k \quad R^n, \; 1 \quad i < j \quad n,$$

where $(e_k)_{k=1,...,n}$ denotes the canonical base of the R-module R^n.

4.9 Remark (Generalization to Submodules). In the general case it is not sufficient to develop Gröbner theory only for ideals I R. It is possible to replace the ideal by a submodule M F of a free R-module F of finite rank. Then one considers Gröbner bases of M with respect to a suitable term ordering. All statements hold analogously for this more general situation, cf. [CLO1998].

5 Flows in Artin Nets

In the present chapter we apply Gröbner theory from Chapter 4 to Petri nets. Analogously to the computation in p/t nets achieved by matrix theory we use Gröbner bases to compute the kernel of the incidence map of Artin nets. We denote by $R = \boldsymbol{Q} [t_1,...,t_m]$ the ring of polynomials in m indeterminates with rational coefficients.

The following proposition compares the kernel of a linear map such as the incidence matrix with another linear map, which describes the lift to a ring of polynomials. The proposition is fundamental for the explicit computation of flows. We prove it for the general submodule case.

5.1 Proposition (Morphisms over Factor Rings). Consider a factor ring

$$A := R / < h_1,...,h_p >$$

with respect to an ideal generated by a family $(h_i)_{i=1,...,p}$ of polynomials and denote by $: R \to A$ the residue map. In order to compute the kernel of a morphism

$$f_A: A^{n_1} \to A^{n_2}$$

between free A-modules of finite rank, one first chooses a matrix

$$(f_{ij}) \in M(n_2 \times n_1, R),$$

such that f_A is represented by the residue classes

$$((f_{ij})) \in M(n_2 \times n_1, A),$$

and then one considers the extended matrix

$$\tilde{f} = \begin{pmatrix} f_{11} & \dots & f_{1n_1} & H & & \\ \dots & & & \dots & \vdots & \in M(n_2 \times (n_1 + pn_2), R) \\ f_{n_2 1} & \dots & f_{n_2 n_1} & & H & \end{pmatrix}$$

$$H = (h_1 \dots h_p) \in M(1 \times p, R),$$

which represents a morphism

$$R^{n_1} \to R^{p n_2} \to R^{n_2}.$$

Now the first n_1 components of a system of generators for

$$\ker \tilde{f} \to R^{n_1} \to R^{p n_2}$$

are mapped by the residue map onto a system of generators of $\ker f_A \subset A^{n_1}$.

Proof. The following diagram commutes by construction

$$\begin{array}{ccc} R^{n_1} & \xrightarrow{f_R := (f_{ij})} & R^{n_2} \\ {}^{1}\downarrow & & \downarrow{}^{2} \\ A^{n_1} & \xrightarrow{f_A = ((f_{ij}))} & A^{n_2} \end{array}$$

with $_i := {}^{n_i}$, i = 1, 2, induced from the residue map. For the general element $_1(x) \in A^{n_1}$ holds:

$$f_A({}_1(x)) = 0 \Rightarrow {}_2(f_R(x)) = 0 \Rightarrow f_R(x) \in < h_1,...,h_p > R^{n_2}$$

$$f_R(x) \in H \begin{pmatrix} \dots & \vdots \\ & \vdots \\ H & \vdots \end{pmatrix} \begin{pmatrix} y_1 & \vdots \\ \dots & \vdots \\ y_{pn_2} & \vdots \end{pmatrix} \text{ for suitable } y_1,...,y_{pn_2} \in R,$$

$$(x, - y)^T \in \ker \begin{pmatrix} f_{11} & \dots & f_{1n_1} & H & & \\ \dots & & & \dots & \vdots & \text{ for suitable } y^T \in R^{p n_2}, \text{ QED.} \\ f_{n_2 1} & \dots & f_{n_2 n_1} & & H & \end{pmatrix}$$

Now we are ready for explicit computations. We apply the results from Chapter 3 and 4 in the submodule case.

5.2 Algorithm (Computation of P-Flows). Consider an Artin net $N = (T, P, C, w^+, w)$ with rational colour algebra A_Q.
Input. Incidence matrix

$$w_P: \; A_Q^{\,n_1} \qquad A_Q^{\,n_2} \text{ of } N,\ n_1 = \operatorname{card} P,\ n_2 = \operatorname{card} T.$$

Output. System of generators of $Z_1(N, A_Q)$.

Remark 3.4: Represent the rational colour algebra $$A_Q = R / < h_1,...,h_p >$$ as quotient of a ring of polynomials $R = \boldsymbol{Q} [t_1,...,t_m]$.
Proposition 5.1: Lift w_P to a matrix (w_{ij}): $R^{n_1} \qquad R^{n_2}$ and introduce the extended incidence matrix $\tilde{w} := \begin{pmatrix} w_{11} & \dots & w_{1n_1} & H \\ \dots & & & \dots \\ w_{n_2 1} & \dots & w_{n_2 n_1} & H \end{pmatrix} \in M(n_2 \times k, R)$ with $k := n_1 + pn_2$, $H = (h_1...h_p) \quad M(1 \times p, R)$.
Algorithm 4.5: Determine a Gröbner base $G = \{ g_1,...,g_s \}$ of the submodule of R^{n_2}, which is generated by the columns $(w_i)_{i=1,...,k}$ of $\tilde{w}$, together with a matrix $A_{wg} \quad M(k \times s, R)$ with $$(g_1...g_s) = (w_1...w_k) A_{wg}.$$
Lemma 4.8: Determine a system of generators $_{ij}$ of the syzygies of the Gröbner base G and define the column matrix $$T := (_{ij}) \quad M(s \times q, R),$$ q cardinality of the system of generators.
Lemma 4.2: Determine a matrix $A_{gw} \quad M(s \times k, R)$ with $$(w_1...w_k) = (g_1...g_s) A_{gw}$$ by computing the reduction modulo G of the elements $w_i \quad R^{n_2}$, $i = 1,...,k$.
Proposition 5.1: The residue classes in A_Q of the first n_1 components of the columns of the matrix $$(1 - A_{wg}A_{gw}, A_{wg}T) \quad M(k \times (k+q), R)$$ generate $Z_1(N, A_Q)$.

Table 2. Algorithm for the computation of generators of $Z_1(N, A_Q)$

The complexity of Algorithm 5.2 has to take into account the size of the net as well as the degrees and the cardinality of a set of polynomials $(h_i)_{i=1,...,p}$, which generate the

defining ideal of the colour algebra. For complexity results from Gröbner theory cf. [BWK1998], Appendix.

5.3 Remark (Tool Support). The tool Macaulay 2 ([GS1996]) supports all steps from Algorithm 5.2. It works for submodules over rings of polynomials with an arbitrary finite number of indeterminates. A single command calculates a system of generators of flows. Moreover the tool serves as a useful help to study fundamental issues of Commutative Algebra.

5.4 Example (P-Flows of Clock Synchronization). The Artin net from Example 3.2 has the rational colour algebra

$$A_{\boldsymbol{Q}} = \boldsymbol{Q}[\,sh_1, sh_2\,] \quad \boldsymbol{Q}[\,t_1, t_2\,] / < t_1^{n_1} \quad 1, t_2^{n_2} \quad 1 > ,$$

cf. Example 3.5. According to Algorithm 5.2 the kernel of the incidence map w_P can be computed by lifting the map w_P to a map $w_{P,R}$ over the ring of polynomials $R = \boldsymbol{Q}[\,t_1, t_2\,]$. In the next step one has to determine the kernel of the extended matrix

$$\tilde{w} = \begin{matrix} t_2 & 1 & 1 & t_2 & 1 & t_1^{n_1} & 1 & t_2^{n_2} & 1 & 0 & 0 \\ 1 & 1 & t_1 & 0 & 0 & 0 & t_1^{n_1} & 1 & t_2^{n_2} & 1 \end{matrix}$$

$$M\,(\,2 \times 8,\, R\,)$$

and to project the generators onto $Z_1(N, A_{\boldsymbol{Q}})$ under the residue map $R \quad A_{\boldsymbol{Q}}$. For this purpose one uses Gröbner theory. The program Macaulay 2, cf. Remark 5.3, computes the result for different values of n_1 resp. n_2. We obtain with respect to the canonical base

$$(p_i)_{i=1,\ldots,4} \text{ of } A_{\boldsymbol{Q}}^4 \quad C_1(N, A_{\boldsymbol{Q}})$$

$$Z_1(N, A_{\boldsymbol{Q}}) =$$

$$\text{span}_{A_{\boldsymbol{Q}}} < {}_1 = \begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}, \; {}_2 = \begin{matrix} 1 & sh_1 \\ 1 & sh_1 sh_2 \\ sh_2 & 1 \\ 0 & \end{matrix}, \; {}_3 = \begin{matrix} 0 \\ S_1 \\ S_1 \\ 0 \end{matrix}, \; {}_4 = \begin{matrix} S_2 \\ S_2 \\ 0 \\ 0 \end{matrix} >.$$

Note: The first two syzygies generate the kernel of the lifted morphism

$$w_{P,R}: R^4 \quad R^2.$$

The generators from [CM1990], Section 5.2, are:

$$Z_1(N, A_{\boldsymbol{Q}}) =$$

$$\text{span}_{A_{\boldsymbol{Q}}} < {}_1 = \begin{matrix} 0 \\ 0 \\ 0 \\ S_2 \end{matrix}, \; {}_2 = \begin{matrix} 0 \\ S_1 \\ S_1 \\ 0 \end{matrix}, \; {}_3 = \begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}, \; {}_4 = \begin{matrix} 0 \\ (sh_2 \quad 1)\, sh_1 \\ sh_2 \quad 1 \\ (sh_1 \quad 1) \end{matrix} >.$$

Both sets of generators transform into each other:

$$\begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{pmatrix} = B \begin{pmatrix} \gamma'_1 \\ \gamma'_2 \\ \gamma'_3 \\ \gamma'_4 \end{pmatrix} \text{ with } B = \begin{pmatrix} S_2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad GL(4 \times 4, A_Q).$$

Note: $S_1 \, sh_1 = S_1$, because $S_1 (sh_1 - 1) = 0$. For an interpretation of the generating P-flows in terms of the original clock synchronization cf. [CM1990], Section 5.2.

5.5 Example (Minimal Set of Generators). We continue with Example 3.2. In general neither Algorithm 4.3 from [CM1990] nor Algorithm 5.2, which is based on Gröbner theory, produces a minimal set of generators of P-flows. Both algorithms produce a family of 4 generators, but 3 generators are already sufficient.
i) To obtain a minimal system of generators we use the Artin factorization of the colour algebra according to Lemma 3.6. Recall from Example 3.5 the factorization

$$t^n - 1 = \prod_{d|n} \Phi_d(t).$$

The local factors of the colour algebra

$$A_Q = \boldsymbol{Q}[sh_1, sh_2] \quad \boldsymbol{Q}[t_1, t_2] / < \Phi_{n_1}(t_1), \Phi_{n_2}(t_2) >$$

correspond bijectively to the maximal ideals of A_Q, which form the set

$$\text{Specm } A_Q = \{ \mu_{d_1,d_2} := < \Phi_{d_1}(t_1), \Phi_{d_2}(t_2) > : d_j \mid n_j \text{ for } j = 1, 2 \}.$$

If we denote a primitive d-th root of unity by

$$\zeta_d := e^{\frac{2\pi i}{d}} \quad \boldsymbol{C}, \; d \quad \boldsymbol{N},$$

then the maximal ideal μ_{d_1,d_2} Specm A_Q determines the local factor

$$k(\mu_{d_1,d_2}) := A_Q / \mu_{d_1,d_2} \quad \boldsymbol{Q}(\zeta_{d_1}, \zeta_{d_2}).$$

ii) Table 3 shows for every local factor the corresponding component of the incidence map, its kernel dimension and the local values of the four generators $\gamma_j \in Z_1(N, A_Q)$, j = 1,...,4.

In general the kernel of the incidence map $w_P(m)$ has two generators, only the distinguished point $m_{1,1}$ needs three generators. We get:

$$Z_1(N, A_Q) = \text{span}_{A_Q} < \gamma_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \; \gamma_3 - \gamma_2 = \begin{pmatrix} sh_1 \, 1 \\ S_1 + sh_1 sh_2 \, 1 \\ S_1 \, sh_2 + 1 \\ 0 \end{pmatrix},$$

$$ {}_4 - {}_2 = \begin{matrix} S_2 + sh_1 & 1 \\ S_2 + sh_1 sh_2 & 1 \\ 1 \quad sh_2 & \\ 0 & \end{matrix} >, $$

$m = m_{d_1,d_2}$	$w_P(m)$	dim ker $w_P(m)$	${}_1(m)$	${}_2(m)$	${}_3(m)$	${}_4(m)$
$d_1 = d_2 = 1$	$\begin{matrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{matrix}$	3	$\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}$	$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 1 \\ 1 \\ 0 \end{matrix}$	$\begin{matrix} 1 \\ 1 \\ 0 \\ 0 \end{matrix}$
$d_1 = 1,$ d_2 1	$\begin{matrix} {}_{d_2} & 1 & 1 & {}_{d_2} & 1 \\ 1 & 1 & 1 & 0 & \end{matrix}$	2	$\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}$	$\begin{matrix} 0 \\ 1 \\ {}_{d_2} \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 1 \\ 1 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$
d_1 1, $d_2 = 1$	$\begin{matrix} 1 & 1 & 1 & 0 \\ 1 & 1 & {}_{d_1} & 0 \end{matrix}$	2	$\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}$	$\begin{matrix} 1 & {}_{d_1} \\ 1 & {}_{d_1} \\ 0 & \\ 0 & \end{matrix}$	$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$	$\begin{matrix} 1 \\ 1 \\ 0 \\ 0 \end{matrix}$
d_1 1 and d_2 1	$\begin{matrix} {}_{d_2} & 1 & 1 & {}_{d_2} & 1 \\ 1 & 1 & {}_{d_1} & 0 & \end{matrix}$	2	$\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}$	$\begin{matrix} 1 & {}_{d_1} \\ 1 & {}_{d_1} {}_{d_2} \\ {}_{d_2} & 1 \\ 0 & \end{matrix}$	$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$	$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$

Table 3. Localization at the Artin factors of A_Q

because for every maximal ideal m Specm A_Q

$$ \ker w_P(m) = \text{span}_{k(m)} < {}_1(m), ({}_3 - {}_2)(m), ({}_4 - {}_2)(m) >. $$

In order to confirm this result, one can check by some cumbersome computations:

$$ {}_2 = [S_1 (1 - S_2) (sh_2^{n_2 - 1} - 1) + S_1 - 1] ({}_3 - {}_2) $$

$$ {}_3 = ({}_3 - {}_2) + {}_2 = S_1 [1 + (1 - S_2) (sh_2^{n_2 - 1} - 1)] ({}_3 - {}_2) $$

$$ {}_4 = ({}_4 - {}_2) + {}_2 = ({}_4 - {}_2) + [S_1 (1 - S_2) (sh_2^{n_2 - 1} - 1) + S_1 - 1] ({}_3 - {}_2). $$

6 Potential Reachability of Markings

Up to now we have studied the kernel of the incidence map, i.e. we computed the flows of a net. In the present chapter we focus on the image of the incidence map, i.e. we compute the potentially reachable markings. Currently this problem has only been solved for p/t nets. The marking equation of a p/t net N with incidence map

$$w_T: C_0(N) \quad C^1(N, \boldsymbol{Z})$$

is the linear equation

$$m_{post} = m_{pre} + w_T(\ ),$$

cf. Remark 2.6 The problem of potential reachability asks for a solution of the corresponding inhomogenous linear equation. Over a field the problem is solved by Linear Algebra: One has to compare the rank of w_T with the rank of the extended matrix, which is bordered by the additional column $m_{post} - m_{pre}$. In the next step one replaces the field by a ring from the more general class of principal ideal domains such as $\boldsymbol{Z}$:

6.1 Definition (Principal Ideal Domain). A ring R is called *principal ideal domain* iff it has no zero-divisors and every ideal is generated by a single element.

For a principal ideal domain one has to consider also the minors of maximal rank.

6.2 Proposition (Inhomogeneous Linear Equations over a Principal Ideal Domain). Denote by A a principal ideal domain and consider a morphism

$$f: V \quad W$$

between free A-modules of finite rank. For a given element w_0 W we have the equivalence:

$$w_0 \quad \text{im } f$$

$$\text{rank } f = \text{rank } (f, w_0) =: r \text{ and } < \text{minors } (r, f) > = < \text{minors } (r, (f, w_0)) >.$$

Here we have used the notation < minors (r, B) > for the ideal of A generated by all minors of rank r of a given matrix B.

The proof makes use of the basic result about modules over principal ideal domains, ([Bou1981], §4, N^o 3, Théorème 1):

6.3 Theorem (Modules over a Principal Ideal Domain). Denote by A a principal ideal domain and consider a pair M F of A-modules. If F is free and M finitely generated, then

M is free.

There exist non-zero elements

$$_i \quad A, \ i = 1,...,r := \text{rank } (M), \text{ with } \ _i \mid \ _{i+1} \text{ for } i = 1,...,r-1,$$

and a basis $(e_i)_{i \in I}$ of F with a subfamily $(e_i)_{i=1,...,r}$, such that the family $(\varepsilon_i e_i)_{i=1,...,r}$ is a basis of M.

The sequence of ideals $< \varepsilon_i >_{i=1,...,r}$ (*invariants of M relative F*) and the module $\mathrm{span}_A < e_i : i = 1,...,r >$ are uniquely determined.

Proof of Proposition 6.2. Set n := rank V and m := rank W. The statement of the proposition is independent from the choice of bases. We apply Theorem 6.3. After choosing suitable A-bases of V resp. W we can assume, that f has the matrix

$$M(f) = \begin{pmatrix} \varepsilon_1 & & & 0 \\ & \ddots & & \vdots \\ & & \varepsilon_r & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix} \quad M(m \times n, A),\ r = \mathrm{rank}\ f,$$

with invariants $< \varepsilon_i >_{i=1,...,r}$ of f(V) relative W. Set

$$w_0 = (w_1 \dots w_m)^T \quad A^m.$$

The rank condition rank f = rank (f, w_0) means, that the extended matrix (f, w_0) has no nonzero minor of rank r+1, which can easily be seen to be equivalent to

$w_i = 0$ for i = r+1,...,m.

The second condition

$$< \text{minors } (r, f) > = < \text{minors } (r, (f, w_0)) >$$

means

$$< \varepsilon_1 \varepsilon_2 \dots \varepsilon_r > \quad < \varepsilon_1 \varepsilon_2 \dots \varepsilon_{i-1} \hat{\varepsilon}_i w_i \varepsilon_{i+1} \dots \varepsilon_r : i = 1,...,r >,$$

(here $\hat{\varepsilon}_i$ denotes deletion of ε_i)

i.e. $w_i \quad < \varepsilon_i >$ for i = 1,...,r.

On the other hand, the existence of a vector $v = (v_1 \dots v_n)^T \quad A^n$ with $f(v) = w_0$ means

$\varepsilon_i v_i = w_i$ for i = 1,...,r and $w_i = 0$ for i = r+1,...,m.

Obviously both sets of conditions are identical, QED.

For a different proof of Proposition 6.2 cf. [Bou1981], §4, exercice 18.

6.4 Theorem (Potential Reachability in p/t Systems). A given marking m_1 is potentially reachable over $\boldsymbol{Z}$ in the p/t system (N, m_0) with incidence map w iff for $m := m_{post} - m_{pre}$ holds

$$\mathrm{rank}\ w = \mathrm{rank}\ (w, m) =: r$$

$$\text{and } < \text{minors } (r, w) > = < \text{minors } (r, (w, m)) >.$$

Proof. The theorem is a special case of Proposition 6.2, QED.

In [DNR1996], Theorem 5.2, the authors prove an analogous criterion for potential reachability of markings over $\boldsymbol{Z}$. They start from duality theory for vector spaces: The image of a morphism is characterized by the kernel of the dual map. For p/t systems the kernel of the dual map is the space of P-flows. Their generalization over $\boldsymbol{Z}$

considers modulo-invariants, which check the dual problem over certain primes. Of course also the method from [DNR1996] relies on Theorem 6.3.

6.5 Example (Nonreachability of a Marking in a p/t Nets). Fig. 1 in [DNR1996] shows a particular Petri net (N, m_{pre}). The authors construct a distinguished marking m_{post}, which satisfies the marking equation over $\boldsymbol{Q}$. Nevertheless, using modulo-invariants they prove that m_{post} is not reachable over $\boldsymbol{Z}$.

Without using modulo-invariants the same result can be obtained from Theorem 6.4: One computes by hand or e.g. by Macaulay 2:

rank w = rank $(w, m_{post} - m_{pre}) = 4$,

but $< \text{minors}(4, w) > = < 2 >$ $< \text{minors}(4, (w, m_{post} - m_{pre})) > = < 1 >$.

7 Outlook and Future Work

In the present paper we have focused on the rational colour algebra of a commutative net. If we multiply a flow with coefficients from $A_{\boldsymbol{Q}}$ with a common denominator of all its components, then we obtain a flow with coefficients from $A_{\boldsymbol{Z}}$. Concerning the question of reachability the situation is quite different: Proving potential reachability of markings over $\boldsymbol{Z}$ is much more difficult than over $\boldsymbol{Q}$. The generalization of Theorem 6.4 to the class of commutative nets will be the subject of further work.

Commutative nets require the commutativity of the colour functions. They form a subclass of homogenous nets, which is computable with Commutative Algebra. From the point of view of general coloured nets this condition is rather restrictive. The next generalization weakens this condition and considers homogeneous nets with a solvable colour algebra. Accordingly one has to generalize the ring of polynomials before applying non-commutative Gröbner theory. For the latest account on Gröbner theory cf. [BW1998] and [BWK1998].

After leaving the domain of linear equations one enters into the realm of linear inequalities. Here one has to introduce a concept of positivity for homogeneous nets e.g., which is indispensable for the token game on any kind of Petri net.

Bibliography

[AM1969] *Atiyah, Michael; Macdonald, I. G.*: Introduction to Commutative Algebra. Addison-Wesley, Reading, Mass. 1969

[Bou1981] *Bourbaki, Nicolas*: Éléments de mathématique. Algèbre, Chapitre 7, Modules sur les anneaux principaux. Masson, Paris 1981

[Bou1972] *Bourbaki, Nicolas*: Elements of mathematics. Commutative Algebra, Chapter I, Flat modules. Hermann, Paris 1972

[BW1998] *Buchberger, Bruno; Winkler, Franz* (Eds.): Gröbner Bases and Applications. London Mathematical Society Lecture Note Series 251. Cambridge University Press 1998

[BWK1998] *Becker, Thomas; Weispfenning, Volker; Kredel, Heinz*: Gröbner Bases: A computational approach to Commutative Algebra. Springer, Berlin et al. 1998
[CLO1998] *Cox, David; Little, John; O'Shea, Donal*: Using Algebraic Geometry. Springer, Berlin et al. 1998
[CM1990] *Couvreur, Jean-Michel; Martinez, J.*: Linear invariants in commutative high level Petri nets. In: *Rozenberg, G.* (ed.): Advances in Petri Nets 1990. Lecture notes in Computer science, vol. 483. Springer, Berlin et al. 1990
[DNR1996] *Desel, Jörg; Neuendorf, Klaus-Peter; Radola, M.-D.*: Proving nonreachability by modulo-invariants. Theoretical Computer Science 153 (1996), p. 49-64
[GS1996] *Grayson, Daniel.; Stillman, Michael*: Macaulay 2. Available at „www.math.uiuc.edu"
[Jen1992] *Jensen, Kurt*: Coloured Petri nets. Basis Concepts, Analysis Methods and Practical Use. Springer, Berlin et al. 1992
[Vas1998] *Vasconcelos, Wolmer*: Computational Methods in Commutative Algebra and Algebraic Geometry. Springer, Berlin et al. 1998

Testing Undecidability of the Reachability in Petri Nets with the Help of 10th Hilbert Problem

Piotr Chrząstowski-Wachtel

Institute of Informatics, Warsaw University, ul.Banacha 2, 02-097 Warszawa, Poland
pch@mimuw.edu.pl

Abstract. The 10th Hilbert problem is used as a test for undecidability of reachability problem in some classes of Petri Nets, such as self-modifying nets, nets with priorities and nets with inhibitor arcs. Common method is proposed in which implementing in a weak sense the multiplication in a given class of Petri nets including PT-nets is sufficient for such undecidability.

Place-transition nets (PT-nets) are considered "the" Petri nets and, when talking about Petri nets without adjectives, one usually means just this class of nets. The decidability of the reachability problem for Petri nets was proved by Mayr and Kosaraju at the beginning of 80's [Mayr 81],[Kosaraju 82]. The decidability result was difficult to prove and was obtained after years of attacking this problem. It became clear that the decidability of reachability holds for PT-nets "on the edge", so it is very probable that when we extend the class of PT-nets by some facilities, we can improve the expressive power of the nets, but the decidability of the reachability problem may be lost.

The expressive power of PT-nets turned out to be too small for many practical purposes. In order to model some phenomena inexpressible in the PT-nets framework, many extensions of PT-nets were proposed. Among them we consider nets with inhibitor arcs, nets with priorities, nets with reset transitions, self-modifying nets and many others. The reachability problem in all the above mentioned classes is undecidable.

When we come to the idea to extend the class of PT-nets, we should consider the decidability of the reachability problem, because if it turns out to be undecidable, the computer-aided analysis of the net behaviour becomes questionable. At least, when the so-called safety properties are considered, they require assurance that no "bad" marking occurs. And when one cannot decide if a given marking is reachable, the possibilities to answer this question are limited.

There are many methods to prove undecidability of the reachability problem. The reduction of known undecidability results in automata theory or logics is a common tool. In this paper a reduction of the 10th Hilbert problem is presented.

The 10th Hilbert problem has been used first probably in [Hack 76], to show some undecidabilities (for instance the problem whether the reachability sets of two given nets are equal). He observed, that one can almost model multiplication

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 268–281, 1999.
© Springer-Verlag Berlin Heidelberg 1999

by a Petri net in the following sense. A net exists with three places p_1, p_2 and p_3 such that if p_1 and p_2 possess x and y tokens respectively, and no tokens are present on the other places of the net at the initial marking, then for all firing sequences, when a marking is reached in which all the places, except for p_3, have no tokens, then *no more* than xy tokens are on p_3, and, moreover, there exists a firing sequence such exactly xy tokens are on p_3.

This was close to a proof of an undecidability of the reachability problem. The missing touch was the absence of an example of a net, in which exactly xy tokens would occur on p_3 in every reachable marking, which empties the other places. This was, as we know now, impossible. Such net does not exist. The reachability problem is decidable for PT-nets.

However, when talking about various extensions of PT-nets, we can come back to this idea. The paper presents a result, which will allow us to check, whether the reachability problem is undecidable in some classes of Petri nets. A sufficient condition for undecidability is somewhat dual to the condition presented above. We will require that *no less* than xy tokens will be reached on p_3, and xy tokens are also reachable for some reachable marking.

Some examples will be presented to illustrate the applicability of the 10th Hibert problem to undecidability of the reachability problem. The paper has some didactic value also, since the formulation (not the proof!) of the 10th Hilbert problem is elementary, and a formulation of the undecidability result in terms of a token game is appealing.

1 Definitions

We adopt a standard notation for PT nets.

A *PT net* is a four-tuple $N = (P, T, Pre, Post)$ where P and T are disjoint and nonempty sets of *places* and *transitions* with $|P|, |T| \in \mathcal{Z}^+$. The *incidence functions* Pre and $Post$ map $(P \times T)$ into $\mathbb{N}$. Nets can be seen, and drawn as weighted bipartite graphs, where elements of P are denoted by circles, while the elements of T — by bars.

A function $M : P \to \mathbb{N}$ is called a *marking*. If $M(p) = 0$ for each p, then M is called a *zero marking*. Markings are represented in vector form. A *PT system* is a tuple (N, M_0) where N is a PT net and M_0 is the *initial* marking. A transition t is *enabled* at M iff for every $p \in {}^\bullet t : M(p) \geq Pre(p, t)$. If t is enabled at M, then t may *fire* yielding a new marking M' given by $M'(p) = M(p) - Pre(p, t) + Post(p, t)$ for each p. The fact that the firing of t changes the marking M into the marking M' is denoted by $M[t\rangle M'$. Also $M[t\rangle$ denotes the marking into which M is transformed by the firing of t. A *firing sequence* from M is a sequence $\sigma = t_1\, t_2 \cdots t_r \in T^*$ such that $M[t_1\rangle M_1[t_2\rangle M_2 \ldots M_{r-1}[t_r\rangle M_r$ for some $M_1, \ldots, M_r$. The set of firing sequences (the *language*) is denoted by $\mathcal{L}$ and the *reachability set* of (N, M_0) is $R = \{M_0[\sigma\rangle \mid \sigma \in \mathcal{L}\}$.

The reachability problem is for a given net system (N, M_0) and a marking M, to decide whether M is in the reachability set of (N, M_0).

Since PT-nets are often too weak for some purposes, many extensions of PT-nets were proposed. Assume for the rest of the paper that all the classes of Petri nets considered in this paper contain PT-nets as special cases. In the definitions that follow, we will present several useful extensions of PT-nets.

Definition 1.1. *A* net with priorities *is a 5-tuple*

$$N_p = (P, T, Pre, Post, Priority),$$

where the sets $P, T, Pre, Post$ *are defined as previously, and* $Priority : T \rightarrow \mathbb{N}$ *is a function assigning a priority to every transition. In case of a conflict between transitions* $t_{i_1}, \ldots, t_{i_k}$*, only a transition with highest priority is enabled and may fire (in case there is more than one transition with highest priority, all of them are enabled, and the conflict is being resolved in a standard way).*

Definition 1.2. *A* net with inhibitor arcs *is a 5-tuple*

$$N_I = (P, T, Pre, Post, Inh),$$

where the sets $P, T, Pre, Post$ *are defined as previously, and* $Inh : P \times T \rightarrow \{0, 1\}$ *is a function defining the so called* inhibitor arcs*. There is an inhibitor arc from* p *to* t *iff* $Inh(p, t) = 1$*. A transition is enabled in a net with inhibitor arc iff for all* $p \in P : M(p) \geq Pre(p, t)$*, and for all* $p \in P : Inh(p, t) = 1 \Rightarrow M(p) = 0$*. So all places connected to* t *by inhibitor arcs must be empty to enable* t*.*

This definition means that if we want t to fire, $Pre(p, t) = 0$ when $Inh(p, t) = 1$. Otherwise the transition t could not be enabled. In other words: If there is an inhibitor arc from p to t, then no other arc should connect p and t. Inhibitor arcs (p, t) can be modeled by ordinary arcs, when p is bounded. But if not, then their presence leads to a different class of nets.

Definition 1.3. A self-modifying net *is a 4-tuple*

$$N_V = (P, T, Pre_V, Post_V),$$

where the sets P *and* T *are defined as above, and the functions* Pre_V *and* $Post_V$ *are not static, as in the case of the PT-nets, but they depend on an actual marking: they map* $P \times T \times M$ *into* $\mathbb{N}$ *for* M *being a marking.*

So the number of tokens that are carried along an arc depends on an actual marking in self-modifying nets.

Definition 1.4. A reset net *is a 5-tuple*

$$N_R = (P, T, Pre, Post, Reset),$$

where Pre and Post are defined as above, and Reset maps $P \times T$ *into* $\{0, 1\}$*. If* $Reset(p, t) = 1$*, then* $Pre(p, t) = 0$*,* $Post(p, t) = 0$*, and the* (p, t) *arc is called the* reset arc*. A transition* t *is enabled, iff* $Pre(p, t) > 0 \Rightarrow M(p) \geq Pre(p, t)$ *and* $Reset(p, t) = 1 \Rightarrow M(p) > 0$*. After firing* t *all ordinary arcs carry as many tokens as in the case of PT-nets, while the reset arcs empty the input reset places.*

Now we present an idea of a computability by Petri nets (Fig.1).

Definition 1.5. *A function* $f : \mathbb{N}^n \to \mathbb{N}$ *is* computable *by a Petri net, if there exists a net* $N = (P, T, Pre, Post)$*, with certain places and transitions:* $p_{x_1}, \ldots, p_{x_n}, start, stop, p_f, progress \in P$*,* $run, finish \in T$ *connected in the following way: (Fig.1):*

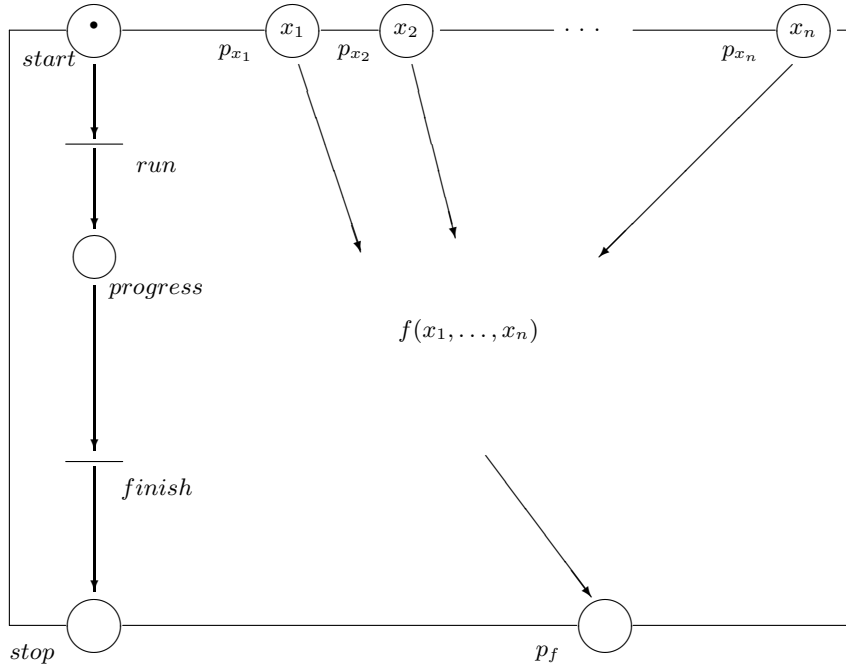


Fig. 1. A net which computes a function. At the initial marking all places are empty except *start* and $p_{x_1}, \ldots, p_{x_n}$. Every place p_{x_i} holds initially x_i tokens. When a marking is reached such that all places but *stop* and p_f are empty, then $f(x_1, \ldots, x_n)$ tokens should be on p_f. The place *progress* is connected by a self-loop with every transition except *run* and *finish*, but to make the picture clean, these arcs are omitted (and will be omitted on pictures that follow).

1. ${}^\bullet start = \emptyset$
2. $Pre(start, run) = 1, Post(progress, run) = 1$
3. $Pre(progress, finish) = 1, Post(stop, finish) = 1$
4. $stop^\bullet = \emptyset, {p_f}^\bullet = \emptyset$
5. $\forall t \in T - \{run, finish\} : Pre(progress, t) = Post(progress, t) = 1$

such that for all initial markings M_0 *satisfying* $\forall 1 \leq i \leq n : M_0(x_i) = a_i$, $M_0(start) = 1$, and $\forall p \in P - \{p_{x_1}, \ldots, p_{x_n}, start\} : M_0(p) = 0$*, when a marking* $M \in [M_0]$ *is reached, the following condition holds:*

$$(M(stop) = 1, \forall p \in P - \{stop, p_f\} : M(p) = 0) \Rightarrow M(p_f) = f(a_1, \ldots, a_n).$$

Since the place *progress* is connected by a self-loop to every transition but *run* and *finish*, the arcs connecting transitions with *progress* will be omitted on all pictures.

The place start, when marked, indicates that the computation has not started yet. When the transition *run* fires, it deposits a token on place *progress*, which indicates that the computation is in progress. Once the transition *finish* is fired, no other transition can fire, because a token is taken from *progress*, and this disables all transitions of the net. At this final stage all places but *stop* and p_f should be empty, and the place p_f should hold exactly $f(x_1, \ldots, x_n)$ tokens.

If we want to reach a marking in which only places *stop* and p_f are marked, then it is necessary to do all the job (emptying the other places) **before** the transition *finish* is fired. To illustrate the idea of computability, let's consider the net from Fig.2. It shows that the addition is computable by Petri nets.

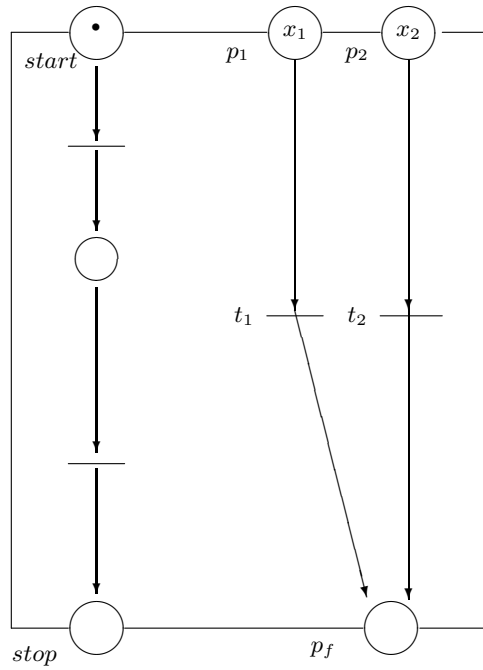


Fig. 2. Addition is computable by PT-nets

The only transition which may fire at the initial marking is *run*. When a token appears on *progress*, one cannot fire *finish* before transition t_1 fires x_1 times and transition t_2 fires x_2 times. Otherwise, the places p_1 and p_2 would

not be empty at the final marking. When this happens, and $x_1 + x_2$ tokens are collected on place p_f, one may fire $finish$ resulting in a final marking with a token on *stop*.

2 Integer Polynomials and the Reachability

The 10th Hilbert problem is to determine, for given polynomial $W(x_1, \ldots, x_n)$ with integer coefficients whether $W(x_1, \ldots, x_n) = 0$ for some integer numbers $x_1, \ldots, x_n$. As it was shown by Matijasevich ([Matijasevich 70]), the 10th Hilbert problem is undecidable.

Let's start with a technical lemma stating that any combination of tokens is easily obtainable on selected places by firing some extra transitions added to a net.

Lemma 2.1. *For a net on Fig.3 every marking is reachable from a zero marking.*

Proof. Assume that the marking M is given. First we fire the transition t_0 $\sum_{p \in P} M(p)$ times, and then depositing as many tokens as we need consecutively on places $p_{x_1}, \ldots, p_{x_n}$, we transfer them from left to right. □

This lemma will be needed in preparing input data $x_1, \ldots, x_n$ for a polynomial to compute its value. The net from Fig.3 will produce the required number of tokens on each place to let the polynomial have the zero value if it is possible at all.

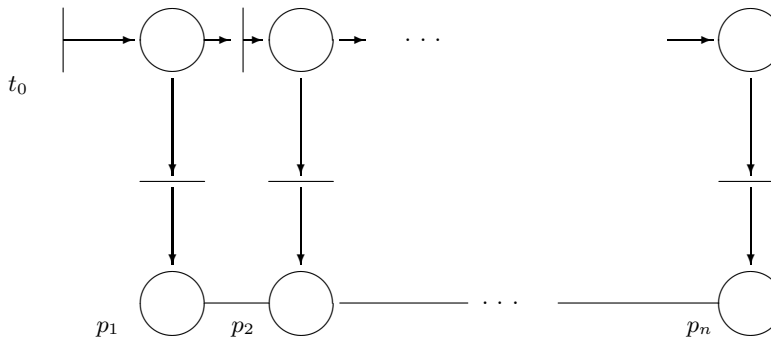


Fig. 3. Every combination of tokens can be generated

Assume now that for every polynomial W over nonnegative integers (the problems with negative arguments will be discussed later) we can construct a net, which for given $x_1, \ldots, x_n$ would compute $W(x_1, \ldots, x_n)$ on a place p_f (in other words, polynomials would be computable in the sense given above). Since,

according to the above lemma, we are able to generate *any* configuration of input $x_1, \ldots, x_n$ tokens on places $p_{x_1}, \ldots, p_{x_n}$, it is enough now to ask, whether a marking M, such that $M(p) = 1$ for $p = stop$ and $M(p) = 0$ for all other places, is reachable, to decide if the equation $W(x_1, \ldots, x_n) = 0$ has an integer solution.

Hence, if a net computing every polynomial in the sense given above existed, then it would mean that the reachability is undecidable — the 10th Hilbert problem would be reduced to the reachability problem.

First we will show a general result concerning all classes of Petri nets containing PT-nets.

Theorem 2.1. *If* $f : \mathbb{N}^2 \to \mathbb{N}$ *and* $g : \mathbb{N}^2 \to \mathbb{N}$ *are computable by Petri nets, then the function* $h(x_1, x_2, x_3) = g(f(x_1, x_2), x_3)$ *is also computable.*

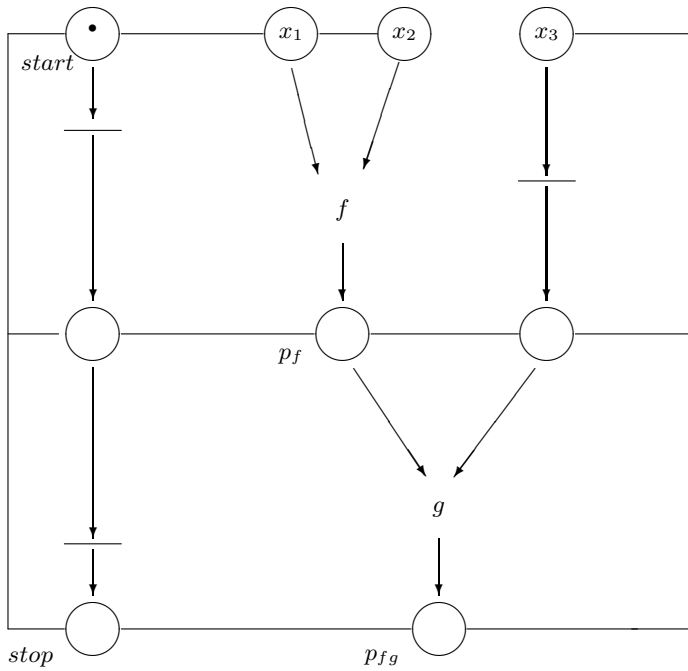


Fig. 4. $g(f(x_1, x_2), x_3)$ is computable if g and f are computable

Proof. Consider the net on Fig.4. The only way to empty all places except *stop* and p_{fg} is to perform a computation of f first, transfer the x_3 tokens to let them become input for g, and then the computation of g does the job.

□

Corollary 2.1. *If multiplication is computable in a class $\mathcal{K}$ of Petri nets, then polynomials with nonnegative coefficients are computable in $\mathcal{K}$.*

Proof. Immediate, since polynomials can be expressed as a composition of multiplications and additions, and addition has been above shown to be computable. □

Theorem 2.2. *If multiplication is computable in a class $\mathcal{K}$ of Petri nets, then the reachability problem is undecidable in $\mathcal{K}$.*

Proof. Assume that the multiplication is computable in $\mathcal{K}$. First we will solve the problem of negative values in the integer domain. The computability was defined above in terms of natural numbers, while the 10th Hilbert problem is stated in the domain of integers.

We will first show that for every polynomial W we can construct a finite set of polynomials $W_1, \ldots, W_M$ for some M such that W has a zero value for some integers $z_1, \ldots, z_n$ if and only if at least one polynomial W_i has a zero value for some nonnegative integers $x_1, \ldots, x_n$. To observe this, it is enough to consider all $M = 2^n$ polynomials resulting from W by substituting for every $I \subset \{1, ..., n\}$ and all $i \in I : -x_i$ in place of x_i into W. Now when a set of integer values $z_1, \ldots, z_n$ is chosen such that $W(z_1, \ldots, z_n) = 0$, then at least the polynomial, which resulted from W by substituting the variables corresponding to negative values of z_i by their negatives, has also a value 0 for nonnegative integers. On the other hand, when one of the polynomials $W_1, \ldots, W_M$ has a value 0 for some nonnegative integers, then by negating the arguments corresponding to the negated variables we will get a solution of $W(z_1, \ldots, z_n) = 0$ in the integer domain.

So from the decidability point of view it is irrelevant, whether we are looking for integer or nonnegative integer solutions.

What about negative coefficients in W? When we know that only nonnegative integers are taken into consideration, we can split the polynomial W into a negative and nonnegative part grouping all the negative coefficients into a polynomial $W^-(x_1, \ldots, x_n)$ and the nonnegative ones into a polynomial $W^+(x_1, \ldots, x_n)$. Since $W^+ - (-W^-) = W$, we can reduce the problem of finding a nonnegative integer solution of W to testing whether two polynomials W^+ and $-W^-$ — both with nonnegative coefficients — have equal values for some nonnegative integers $x_1, \ldots, x_n$.

This can easily be done by the net from Fig.5.

A copy of $x_1, \ldots, x_n$ is done to prepare the same input for W^+ and W^-. The presented net reaches the *stop* phase with $M(p) = 0$ for all $p \in P - \{stop\}$ if and only if W^+ and W^- have the same value for some nonnegative $x_1, \ldots, x_n$.

So when multiplication is computable in a considered class, we are provided with a method to check the existence of an integer solution of every equation $W(x_1, \ldots, x_n) = 0$ in terms of the reachability problem. This concludes the proof □

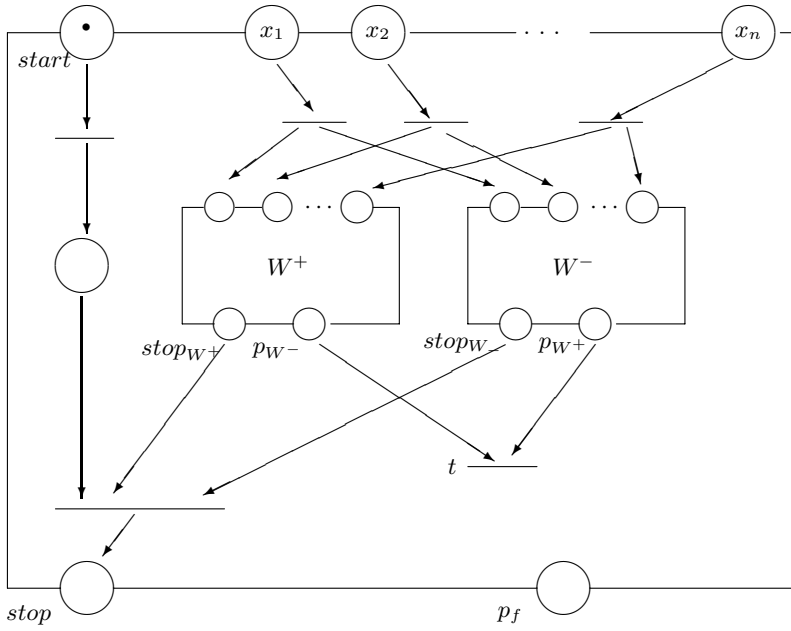


Fig. 5. Testing if $W^+(x_1, \ldots, x_n) = W^-(x_1, \ldots, x_n)$ for given $x_1, \ldots, x_n$.

Now we know that since the reachability problem is decidable for PT-nets, the multiplication is not computable by PT-nets.

A characterization of functions computable by PT-nets was presented in [Tarlecki 83]. The linear functions $f(x_1, \ldots, x_n) = k_1x_1 + \cdots, +k_nx_n$ for non-negative integers $k_1, \cdots, k_n$ are the only functions computable by PT-nets.

Is it a dead end? Fortunately not. To make the 10th Hilbert problem applicable, let's first observe that there are classes of Petri nets, for which the multiplication is computable. The following two results are well established, but the original proofs did not use the 10th Hilbert problem to determine undecidability results.

Proposition 2.1. *The reachability problem is undecidable in the class of self-modifying nets.*

Proof. Consider the net from the Fig.6.

Observe that to empty the net interior one must fire x times the transition t. Each time t is fired, y tokens are transmitted to p_f. Finally, the $finish$ transition clears the tokens from the p_y place resulting in xy tokens on p_f. □

Proposition 2.2. *The reachability problem is undecidable for nets with priorities.*

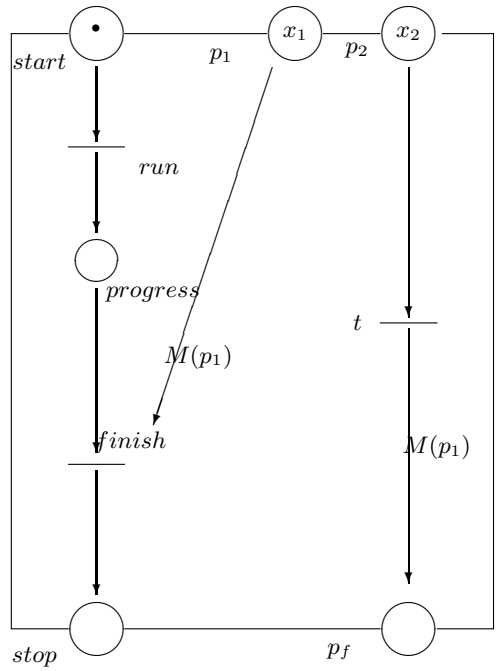


Fig. 6. The reachability is undecidable for self-modifying nets

Proof. Consider the net from the Fig.7. Assume that each transition t_i has a priority i. Observe that since all possibly enabled transitions will be in conflict after a token appears on place *progress*, the priorities determine the order in which the transitions are executed. Immediately after *run* is fired, only t_2 is enabled (if there are tokens on p_y)[1]. When one token from p_y is put on p_2, t_4 fires x times making a copy of x on p_1. Now only t_3 is enabled, so it fires, and a token is put on p_3 enabling t_6. This transition fires x times returning the tokens on place p_x, and now t_5 is the only enabled transition, and fires removing the token from p_3. We have executed the sequence $t_2 t_4^x t_3 t_6^x t_5$, and the marking we have reached differs from the previous one in such a way that the place p_y has one token less, and the place p_f has x tokens more. The above sequence is repeated $y-1$ times in total, resulting in a marking in which 1 token is on p_y, $x(y-1)$ tokens on p_f and x on p_x. In the last round the sequence $t_2 t_4^x t_3$ is fired, after which t_6 is no longer enabled, since no tokens are on p_y. So t_5 fires, after which $t_1 x$ empties the place p_1, and the transition *finish* is ready to terminate the whole process leaving xy tokens on p_f. This schedule is purely deterministic. □

[1] In fact if $y = 0$, then the net does not work well: no transition can empty the place p_x. To make the proof work also for the case $y = 0$ one should add an extra transition with priority $\frac{1}{2}$, which would take tokens from p_x. This transition would never fire in case $y > 0$.

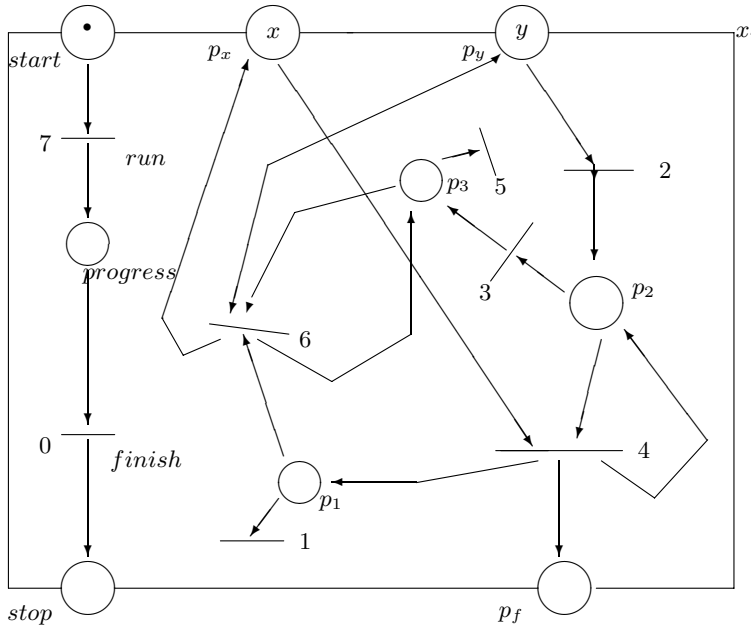


Fig. 7. The reachability is undecidable for nets with priorities

3 Weak Computability is also Sufficient

We will strengthen the result from the previous section. Hack has observed that multiplication is computable in a weaker sense for PT-nets. The weak computability differs from the computability in the following sense. Instead of requiring that exactly $f(x_1, \ldots, x_n)$ tokens are on p_f when a token reaches *stop* and all other places are unmarked, we require only that **no more** than $f(x_1, \ldots, x_n)$ tokens are on p_f, and, moreover that a firing sequence exists, which produces **exactly** $f(x_1, \ldots, x_n)$ tokens on p_f. We propose a name *bottom-computability* for this kind of computability.

The following proposition reminds the Hack's observation.

Proposition 3.1. *Multiplication is bottom-computable in PT-nets.*

Proof. Consider the net from the Fig.7. If we forget about priorities and let the transitions fire in any order, then we will see that the schedule with priorities produces the maximum number of tokens on p_f. Each other schedule produces less tokens (for instance firing less times t_4 after each firing of t_2). □

We will introduce now a somewhat dual notion: *top-computability*. We say that a function f is top-computable, if **no less** than $f(x_1, \ldots, x_n)$ tokens are on the place p_f, when a marking is reached in which $M(stop) = 1$, and all other

places are unmarked, and, as previously, we require that a firing sequence exists, which produces **exactly** $f(x_1, \ldots, x_n)$ tokens on p_f.

Clearly when a function is computable by Petri nets, it is both bottom- and top-computable. The following theorem states that the reverse is also true.

Theorem 3.1. *If a function $f(x_1, \ldots, x_n)$ is bottom-computable and top-computable in a class $\mathcal{K}$ of Petri nets, then f is computable in $\mathcal{K}$.*

Proof. Consider the net from the Fig.5, and interpret the two subnets presented on the picture as two nets, which bottom-compute f on place p_{W^-} and top-compute f on place p_{W^+}. In order to empty the interior of the net one must obtain the same number of tokens on places p_{W^-} and p_{W^+}. Add also to the net one arc leading from t to p_f. Since no more than $f(x_1, \ldots, x_n)$ tokens can appear on p_{W^-}, while no less on p_{W^+}, to empty the interior of the net, one must produce exactly $f(x_1, \ldots, x_n)$ tokens both on p_{W^-} and p_{W^+}. Such firing sequences exist from the definition. In this case transfering the tokens to p_f by firing several times t we obtain exactly $f(x_1, \ldots, x_n)$ tokens as the only possibility to exit the computation with the interior unmarked. □

This result enables us to state the following stronger condition for undecidability of the reachability in classes of Petri nets containing PT-nets.

Theorem 3.2. *If in a class $\mathcal{K}$ of Petri nets containing PT-nets multiplication is top-computable, then the reachability problem is undecidable in $\mathcal{K}$.*

Proof. As for PT-nets multiplication is bottom-computable (proposition 3.1), the result comes immediately from theorems 2.2 and 3.1. □

We will use the above theorem to provide a new proof of another known undecidability result.

Proposition 3.2. *The reachability problem is undecidable for nets with inhibitor arcs.*

Proof. Consider the net from the Fig.8. This time we will try to put *as few tokens as possible* on place p_f. The main problem here is to remove tokens from p_y. The only transition that can do this is t_2, but it is connected by an inhibitor arc to p_x, so it cannot fire unless p_x is empty. After the transition *run* fires, only t_1 is enabled. We remove tokens from p_x by firing x times t_1, and, as a side effect, deposit x tokens on p_f. Now we could run the cycle t_3t_1 for several times, but this would mean more tokens on p_f, so we don't do it.

Instead, we fire t_2. A token moves from p_1 to p_2, and now in order to fire t_2 again we must empty the place p_3. This can be done only by firing x times transition t_3. Now transition t_4 can fire returning the token from p_2 back to p_1. This completes the cycle. We have now one token less on p_y, and x tokens more on p_f, while other places are marked the same, as just after *run* had fired. We repeat $y - 1$ times in total this cycle $t_1^x t_2 t_3^x t_4$, and in the last run after the sequence $t_1^x t_2$ is fired, we clean the net by firing t_5 and t_6 and leaving *at least* xy tokens on p_f. □

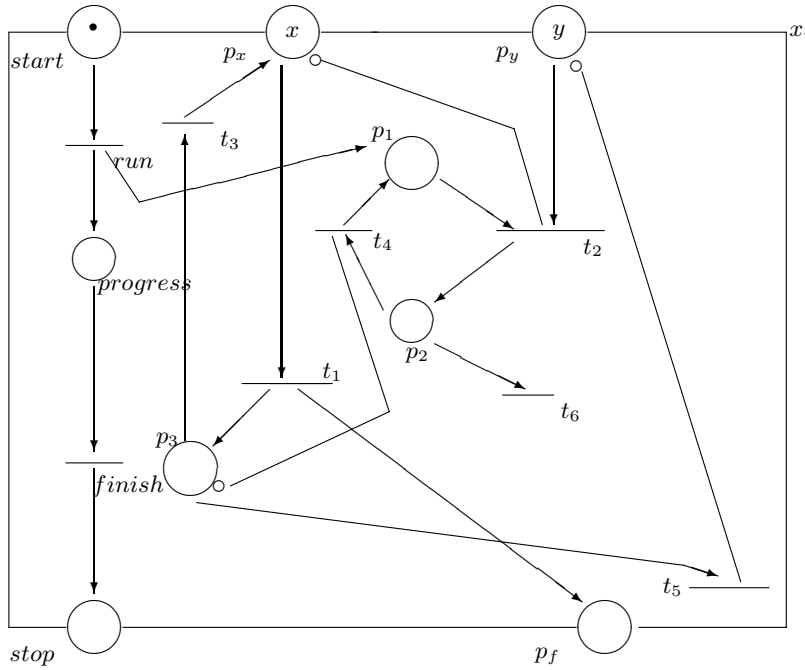


Fig. 8. The reachability is undecidable for nets with inhibitor arcs

In the net used in the above proof, three arcs are present. One of them (connected to t_5) can be easily eliminated at the cost of some slight complications of the net structure — it is preserved here for the clarity of presentation. In particular the mentioned net with two inhibitor arcs provides a proof for the undecidability of reachability in nets with at most two inhibitor arcs: it is not difficult to construct the polynomial computation in a way in which the above net works as a procedure and can be re-used. However, the author was unable to construct a net with 1 inhibitor arc, which would compute multiplication in a weak sense. If such a net was constructed, this would negatively solve the open yet problem if the reachability is decidable for nets with exactly one inhibitor arc ([Kindler 98]).

It is worthwile to mention that theorem 3.2 can be slightly strengthened.

Theorem 3.3. *Let $\mathcal{K}$ be a class of Petri nets containing PT-nets. If squaring is top-computable in $\mathcal{K}$, then the reachability problem is undecidable in $\mathcal{K}$.*

Proof. First observe that the function $f(x) = x/2$ for even x is computable by Petri nets. Just one transition suffices, which takes 2 tokens from p_x and deposits 1 token on p_f. So if the function $f(x) = x^2$ is top-computable, then it is computable by Petri nets. And since $xy = \frac{(x+y)^2}{2} - \frac{x^2+y^2}{2}$ holds for all x and

y, and all the functions on the right hand of the equality are computable, their composition is computable as well.

One should mention that constructing a net example is not always as simple as in the presented classes of nets. Probably for some classes of nets it is just impossible to apply these results. The author was unable to construct such example for the reset nets, for which the undecidability of reachability is known [DFSch 98].

4 Conclusions

Some sufficient conditions were presented, which enable us to decide undecidability of the reachability problem in several classes of Petri nets. The applicability of them was illustrated on several well investigated classes of nets. Although no fresh discoveries were made concerning the undecidabilities, a common framework has been used for the undecidability proofs in these classes.

Since the 10th Hilbert problem has an elementary formulation (though far from being elementary proof), these results has a didactic value, since they allow to prove the undecidability results without the necessity to refer to some advanced notions from logics, automata theory or the theory of languages.

The criterion for undecidability of the reachability is so simple that it often allows to construct a net quite easily, when an extension of PT-nets is investigated.

References

DFSch 98. C.Dufourd, A.Finkel, Ph.Schnoebelen, *Reset nets: between decidability and undecidability*, Proceedings ICALP 98, Aalborg (1998)

Hack 74. M.Hack, *Decidability question for Petri nets*, PhD thesis, Department of Electrical Engineering, MIT 1974.

Hack 76. M.Hack, *The equality problem for vector addition systems is undecidable*, Th. Comp. Sc. 1976, 2, pp 77-95 (1976)

Kindler 98. E.Kindler, *Private communication.*

Kosaraju 82. S.R.Kosaraju, *Decidability of Reachability in Vector Addition Systems*, In Proc. of the 14th Annual ACM Symp. on Theory of Computing, San Francisco, May 5-7, 1982, pages 267-281 (1982)

Matijasevich 70. Y.Matijasevich, *Enumerable sets are Diophantine*(in Russian), Dokl. Akad. Nauk SSSR, 191, pp 279-282 (1970). Translation in: Soviet Math. Doklady, 12 pp 249-254, (1971)

Mayr 81. E.Mayr, *An algorithm for the general Petri net reachability problem.* Proceedings of the 13th Ann. ACM Symposium on Theory of Computing, May 11–13, 1981 (Milwaukee, WI), pp. 238–246. Also: SIAM Journal on Computing **13**, 3, pp. 441–460.

Tarlecki 83. A.Tarlecki, *An obvious observation on functions computable by Petri Nets*, SIG Petri Nets and Related Models, Newsletter 13 (February 1983)

Net Theory and Workflow Models

Giorgio De Michelis

DISCO - University of Milano “Bicocca”
Via Bicocca degli Arcimboldi 8, I-20126 Milano – Italy
email: gdemich@dsi.unimi.it

Abstract.

Petri Nets have been popular among the developers of workflow management systems for more than twenty years [5]: even if we do not consider the early work of Petri himself and Anatol Holt on modelling procedures with Petri Nets, Paul Zisman and Clarence Ellis adopted Petri Nets for modelling workflows in the late seventies. From those early years, there has been a growing amount of proposals adopting different classes of Petri Nets as the modelling framework of a workflow management system. The main reasons of the popularity of Petri Nets are the following:

- they allow to give to workflow models a univocal non ambiguous semantics;
- they have an easy to read graphical representation;
- they may support a hierarchy of abstraction levels;
- they are executable models, well suited for both simulation and software specifcation.

Today workflow technology, even if it is still considered as a hot technology whose success is imminent, has not yet gained large shares of the computer-based applications’ market. Trying to explain this apparent paradox, all its components (the workflow engine, the workflow models and the workflow design environment) have been deeply discussed [1]. Workflow models (and within them, also Net models) have been criticized for the following reasons:

- they are too rigid since, by imposing an explicit flow of actions, they hinder users in overcoming breakdowns;
- they are too complicated, since their design requires a professional expert and cannot be performed by users themselves.

On the contrary, it has been claimed that, in order to become the kernel of really usable workflow management systems, workflow models should have the following features:

- they should be easily modifiable, supporting both the automatic verification of change correctness and safe change enactment on the ongoing instances [6];
- they should support exception handling, allowing users to follow exceptional paths on the basis of the policy of their organization, without making models too complicated;

S. Donatelli, J. Kleijn (Eds.): ICATPN’99, LNCS 1639, pp. 282283, 1999.
© SpringerVerlag Berlin Heidelberg 1999

- they should offer multiple diverse views of the workflow to their diverse users (perfomers, managers, customers, designers).

I claim that Net Theory offers a very powerful platform for satisfying the above requirements, going far beyond the modelling capabilities that have been exploited in the workflow models developed up to now. In particular, I think that process extensions, net morphisms and synthesis algorithms provide powerful mathematical tools for dealing with the above requirements. In the Milano workflow management system [3] we have used a subclass of Elementary Net Systems [7], that has efficient algorithms for all the above services, for creating simple workflow models, that are easily changeable, allow exceptional paths and support a variety of views on the workflow [2]. Even if we claim that workflow management systems do not need more powerful models, since simplicity is a positive attribute for them (at least with respect to a large class of workflows), there are several other classes of processes (production and logistics processes, juridical processes, ...), which may need modelling capabilities that go beyond the subclass of Elementary Net Systems we have chosen for the Milano Workflow Management System. The search for classes of Net Systems well supported by efficient algorithms is therefore open [4].

References

1. K. R. Abbott, S. K. Sarin: Experiences with Workflow Management: Issues for the Next Generation. In R. Furuta and C. Neuwirth (eds.): CSCW'94. *Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, October 22-26, 1994*. New York, NY: ACM Press, pp. 113-120, 1994.
2. A. Agostini, G. De Michelis: Simple Workflow Models. In *Workflow Management: Net-based Concepts, Models, Techniques and Tools*, Computing Science Report 98/07, Eindhoven, The Netherlands: Eindhoven University of Technology, pp. 146-164, 1998.
3. A. Agostini, G. De Michelis: A light workflow management system using simple process models. Computer Supported Cooperative Work (CSCW) The Journal of Collaborative Computing, 1999, (to appear).
4. E. Badouel, Ph. Darondeau: Theory of Regions. In W. Reisig, G. Rozenberg (eds.): *Lectures on Petri Nets I: Basic Models*. LNCS 1491, Berlin, Germany: Springer Verlag, pp. 529-586, 1998.
5. G. De Michelis, C. A. Ellis: Computer Supported Cooperative Work and Petri Nets. In W. Reisig, G. Rozenberg (eds.): *Lectures on Petri Nets II: Applications*, LNCS 1492, Berlin, Germany: Springer Verlag, pp. 125-153, 1998.
6. C.A. Ellis, K. Keddara, G. Rozenberg: Dynamic Change within Workflow Systems. In N. Comstock, C. Ellis (eds.): COOCS'95. *Proceedings of the Conference on Organizational Computing Systems, Milpitas, CA, August 13-16, 1995*. New York, NY: ACM Press, pp. 10-21, 1995.
7. G. Rozenberg, J. Engelfriet: Elementary Net Systems. In W. Reisig, G. Rozenberg (eds.): *Lectures on Petri Nets I: Basic Models*. LNCS 1491, Berlin, Germany: Springer Verlag, pp. 12-121, 1998.

Concurrent Implementation of Asynchronous Transition Systems

Walter Vogler*

Institut für Informatik, Universität Augsburg, Germany
vogler@informatik.uni-augsburg.de

Abstract. The synthesis problem is to decide for a deterministic transition system whether a Petri net with an isomorphic reachability graph exists and in case to find such a net (which must have the arc-labels of the transition system as transitions). In this paper, we weaken isomorphism to some form of bisimilarity that also takes concurrency into account and we consider safe nets that may have additional internal transitions. To speak of concurrency, the transition system is enriched by an independence relation to an asynchronous transition system.
For an arbitrary asynchronous transition system, we construct an ST-bisimilar net. We show how to decide effectively whether there exists a bisimilar net without internal transitions, in which case we can also find a history-preserving bisimilar net without internal transitions. Finally, we present a construction that inserts a new internal event into an asynchronous transition system such that the result is history-preserving bisimilar; this construction can help to find a history-preserving bisimilar net (with internal transitions).

1 Introduction

One methodology for the design of asynchronous circuits takes a transition system (whose arcs are labelled with what we call events) as specification of the desired behaviour, gives it a distributed implementation as a safe Petri net and transforms the latter stepwise into a circuit, see e.g. [CKLY98]; [Yak98] gives in detail a practical example for such a development. In the synthesis of the net, it is desirable that each event of the transition system corresponds to a unique transition of the net: as [Yak98] shows, the transformation of the net may involve event refinement, which is much easier when each event is represented by one transition; only with this property, it is e.g. easy to refine an event e such that each occurrence of e in a run of the original net is replaced alternatingly by an occurrence of e_1 or one of e_2 in a run of the refined net. Also some existing procedures for efficient direct compilation of a net into an asynchronous circuit rely on this property.[1] Since we desire this property, it is natural that we restrict attention to deterministic transition systems.

A pivotal contribution to the synthesis problem is the theory of regions, see e.g. [NRT92]; it allows a characterization of those transition systems TS for

* Work partially supported by the DFG-project 'Halbordnungstesten'.
[1] Thanks go to Alex Yakovlev for pointing this out to me.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 284–303, 1999.
© Springer-Verlag Berlin Heidelberg 1999

which an elementary Petri net exists whose reachability graph is isomorphic to TS. Elementary nets are (almost) the same as safe nets without loops, having the nice feature that independence of transitions corresponds exactly to 'diamonds' in the transition system. But [PKY95] points out, that loops are very natural in particular in the context of circuits and allow to implement additional transition systems; the theory of regions is extended accordingly (almost) to general safe nets, which can be seen as a specialization of the parametric results in [DS93].

[CKLY98] weakens the requirements of [NRT92] and shows that, for each transition system TS satisfying the weaker requirements, there exists an elementary net whose reachability graph is bisimilar to TS. For nets or transition systems (without internal events), bisimilarity and language equivalence coincide; based on [HKT92], language equivalent realization of transition systems is studied in [BBD95] for bounded nets and in [Dar98] for unbounded nets.

[Yak98] is confronted with a transition system that cannot be realized by a net in any of these approaches; since the reaction to such a situation can hardly be simply to give up, he first inserts an internal event into the transition system 'in a harmless way', i.e. he achieves in his example an implementation with a net that has an additional internal transition. One can see that this net (i.e. its reachability graph) is bisimilar to the original transition system; implementation of transition systems in this sense is the topic of the present paper.

Thus, we are given a deterministic transition system TS as a specification of some behaviour and we want to find a (general) safe net whose transitions are the events of TS and possibly some additional internal events, and whose reachability graph is (weakly) bisimilar to TS. This ensures (but see below) that the net has essentially the desired behaviour; also, properties formulated e.g. in Hennessy-Milner logic and checked for the transition system will also hold for the net, see [Yak98] for an informal check of this type. Note that internal events could lead to a new and usually unwanted behaviour that is ignored by bisimulation, namely divergence, i.e. infinite internal computation; hence, we additionally require the net to be divergence-free.

It should be mentioned that it is easy to turn an arbitrary transition system into a Petri net when labelling of the transitions is allowed. Furthermore, constructions are known – see e.g. [BDKP91] – that turn a labelled Petri net into one where there are additional internal transitions, but otherwise each label occurs only once; the latter is the kind of net we are looking for. To the best of the author's knowledge, the known constructions either do not give a bisimilar net or introduce divergence. We will improve on this.

In fact, it is not too difficult to find a bisimilar divergence-free implementation for each transition system, but it turns out to be completely sequential. This is undesirable e.g. for performance reasons; hence, an additional requirement is to preserve concurrency – for which concurrency must be specified in the first place. We will therefore in fact start from an asynchronous transition system (ATS), which is a deterministic transition system with an additional independence relation on the events. In [DS93,NW95], asynchronous and similar transition systems are related to nets by characterizing those ATS for which a

net exists whose reachability graph (with independence relation) is isomorphic to the ATS; instead of requiring isomorphism, we want to compare the behaviour of an ATS and a net.

Preservation of concurrency could mean that the net should have the same step sequences or partial order semantics as the ATS, where the latter semantics could be defined via Petri net processes or equivalently as (Mazurkiewicz) traces; see e.g. [NW95], also for ATS. Or one could combine this with bisimilarity and require step or history-preserving bisimilarity; for our first result, we will consider something in between: ST-bisimilarity, which combines bisimulation with a partial order semantics based on so-called interval orders, see e.g. [Vog92]. We will show that each ATS (with a weak requirement for independence) can be implemented by an ST-bisimilar divergence-free safe net.

Then, we will consider ATS with the usual strong requirement for independence; we will show how to decide whether for such an ATS there exists a safe net *without* internal events with a bisimilar reachability graph, and we will prove that one such net is in fact history-preserving bisimilar to the ATS. Finally, we will mention an ATS-modification that sometimes helps to find a history-preserving bisimilar divergence-free safe net *with* internal events; a special case is the modification used in [Yak98] and mentioned above.

Two other very interesting contributions to the synthesis problem must be mentioned. In [Sun98, Chapter 5], the specification of the desired behaviour is given as a temporal logic formula, and it is shown how to decide whether there exists a safe net with possibly some additional internal events meeting the specification. [Dar98] generalizes the synthesis problem in two other ways: firstly, two regular languages are given and a (possibly unbounded) net is sought for with a language between the given ones; secondly, the problem of realizing a deterministic context-free language with a net is considered.

2 Petri Nets and ST-Bisimulation

In this paper, a *safe Petri net* N (or just a *net*) is a tuple (S, E_v, E_i, F, M_N) satisfying a number of requirements explained in the following. S, E_v and E_i, are finite disjoint sets of *places* and *visible* and *internal events*; thus, we call the elements of $E = E_v \cup E_i$ events instead of transitions. N is called *visible* if E_i is empty. $F \subseteq S \times E \cup E \times S$ is the set of *arcs* (which all have weight 1), and M_N is the *initial marking*, which is as any *marking* a subset of S. When we introduce a net N or N' etc., then we assume that implicitly this introduces its components S, E, E_v, ... or S', E', E'_v, ..., etc. and similarly for other tuples later on.

For each $x \in S \cup E$, the *preset* of x is ${}^\bullet x = \{y \mid (y,x) \in F\}$ and the *postset* of x is $x^\bullet = \{y \mid (x,y) \in F\}$. These notions are extended pointwise to sets, e.g. ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$. If $x \in {}^\bullet y \cap y^\bullet$, then x and y form a *loop*.

- An event a is *enabled* under a marking M, denoted by $M[a\rangle$, if ${}^\bullet a \subseteq M$. If $M[a\rangle$ and $M' = (M \setminus {}^\bullet a) \cup a^\bullet$, then we denote this by $M[a\rangle M'$ and say that a can *occur* or *fire* under M yielding the marking M'. (This rule is in fact

a bit unusual, since we simply take the union of the possibly overlapping $M \setminus {}^\bullet a$ and $a^\bullet$. Our rule coincides with the usual one for the nets we will consider in the following.)

- The definition of enabling ($M[w\rangle$) and occurrence ($M[w\rangle M'$) is extended to sequences w as usual. If w is enabled under the initial marking, it is called a *firing sequence*. A marking M is called *reachable* if $\exists w \in E^* : M_N[w\rangle M$. The net is *safe* if for all reachable markings M and events a, $M[a\rangle$ implies $(M \setminus {}^\bullet a) \cap a^\bullet = \emptyset$.

General Assumption All nets considered in this paper are safe and have only events with nonempty presets. For convenience, we also assume that for each event a there is some reachable marking that enables a.

It is obvious how to generalize the firing rule to infinite sequences. It is usually desirable that a net be *divergence-free*, i.e. that no reachable marking enables an infinite sequence of internal events. Next, we lift the enabledness and firing definitions to the level of visibility:

- A sequence $v \in E_v^*$ is *visibly enabled* under a marking M, denoted by $M[v\rangle\rangle$, if there is some sequence $w \in E^*$ with $M[w\rangle$ such that v is obtained from w by deleting all internal events. If $M = M_N$, then v is called a *visible firing sequence*.

To each net N we associate an *independence relation* $I(N)$ on its events, where $a\, I(N)\, b$ if ${}^\bullet a \cup a^\bullet$ and ${}^\bullet b \cup b^\bullet$ are disjoint. Note that $I(N)$ is irreflexive and symmetric. For a marking M, $\mu \subseteq E$ is an *M-step* if the events in μ are enabled under M and pairwise independent; in this case, the events can fire in any order under M and, intuitively, also simultaneously.

We are interested in behaviour notions capturing choice *and* concurrency in a strong sense, hence in variants of bisimulation that also consider concurrency. (For the basic bisimulation, see the next section.) One such variant is ST-bisimulation; its key idea is that the firing of a visible event a consists of a beginning a^+ and an end a^-, where a^+ checks the enabledness of a and consumes its input, while a^- produces the output. Thus, concurrency in the sense of overlapping occurrences can be observed for visible events – while internal events cannot be observed at all. This is a stronger notion of concurrency than e.g. steps; it corresponds to a partial order semantics that is weaker than causality as captured by net processes, but is instead based on so-called interval orders (see e.g. [Vog92]) and suitable to judge temporal efficiency when events take time, see [Vog95].

If events have a beginning and an end, a system state cannot adequately be described by a marking alone; instead, it consists of a marking together with some events that have started, but have not finished yet, and it is called an ST-marking. In the corresponding firing rule, the preset of a starting event is usually subtracted from the marking immediately [GV87]; for compatibility with the next section, we introduce an alternative but equivalent formulation.

- An *ST-marking* (M, μ) of a net N consists of a reachable marking M and an M-step $\mu \subseteq E_v$. The *initial ST-marking* is $(M_N, \emptyset)$.
- For a visible event a, we write $(M, \mu)[a^+\rangle(M, \mu \cup \{a\})$ if $M[a\rangle$ and a is independent to all $b \in \mu$ – which in particular implies $a \notin \mu$. For a visible event a, we write $(M, \mu)[a^-\rangle(M', \mu \setminus \{a\})$ if $a \in \mu$ and $M[a\rangle M'$. Finally, for an internal event c, we write $(M, \mu)[c\rangle(M', \mu)$ if c is independent to all $b \in \mu$ and $M[c\rangle M'$. Note that in all three cases the pair reached is an ST-marking again.
- We again extend this definition to sequences and, by suppressing internal events, to visible sequences.

Nets N and N' with the same visible events are *ST-bisimilar*, if there is an *ST-bisimulation* between them, i.e. a relation $\mathcal{B}$ between their ST-markings with:

1. $\mathcal{B}$ relates the initial ST-markings.
2. If $a \in E_v$, $c \in E_i$ and $(M, \mu)\mathcal{B}(M', \mu)$ (with the same μ), then we have:
   (a) $(M, \mu)[a^+\rangle(M, \mu \cup \{a\})$ implies that for some M'' $(M', \mu)[a^+\rangle\rangle(M'', \mu \cup \{a\})$ and $(M, \mu \cup \{a\})\mathcal{B}(M'', \mu \cup \{a\})$
   (b) $(M, \mu)[a^-\rangle(M_1, \mu \setminus \{a\})$ implies that for some M_1' $(M', \mu)[a^-\rangle\rangle(M_1', \mu \setminus \{a\})$ and $(M_1, \mu \setminus \{a\})\mathcal{B}(M_1', \mu \setminus \{a\})$
   (c) $(M, \mu)[c\rangle(M_1, \mu)$ implies that $(M', \mu)[\rangle\rangle(M_1', \mu)$ and $(M_1, \mu)\mathcal{B}(M_1', \mu)$ for some M_1'
3. vice versa

ST-bisimulations do not consist of pairs $((M, \mu), (M', \mu'))$ for general labelled nets; instead, there is an additional component that matches for each visible label a the a-labelled transitions in μ to the a-labelled transitions in μ'. This is not necessary in our setting, since here a step can contain at most one a – whereas in general there can be several a-labelled transitions in a step.

3 Asynchronous Transition Systems and History-Preserving and ST-Bisimulation

An asynchronous transition system (an ATS) A (and, more generally, a weak asynchronous transition system, a wATS) is a tuple (Q, E_v, E_i, T, q_0, I) satisfying a number of requirements explained in the following. Q is the finite set of *states* containing the *initial state* q_0; E_v and E_i are finite disjoint sets of *visible* and *internal events* and I is the irreflexive and symmetric *independence relation* on $E = E_v \cup E_i$; events are *dependent* if they are not independent. A is called *visible* if E_i is empty. We speak of a *transition system*, if we are not interested in I.

The transition relation T is a partial function from $Q \times E$ to Q, i.e. A is deterministic over the alphabet E. We say a is *enabled* under q or can *occur* from q and write $q \xrightarrow{a}$, if T is defined for (q, a); we write $q \xrightarrow{a} p$, if $T(q, a) = p$, and speak of an (a-labelled) *arc* from q to p; q and a form a *loop* in A if $q \xrightarrow{a} q$. Similarly to the last section, $\xrightarrow{a}$ is generalized to $\xrightarrow{w}$ for $w \in E^*$ (asserting the existence of a w-labelled *path*); if $q_0 \xrightarrow{w}$, then w is called an *occurrence sequence*

of A. Also in direct analogy to the last section, we define *divergence-freeness* of wATS. We write $q \stackrel{v}{\Rightarrow} q'$ if $q \stackrel{w}{\rightarrow} q'$ and v is the sequence of visible events obtained from w by deleting all internal events. We require that all states of A are *reachable*, i.e. for all $q \in Q$ there is some $w \in E^*$ with $q_0 \stackrel{w}{\rightarrow} q$.

For $q \in Q$, $\mu \subseteq E$ is a *q-step* if the events in μ are enabled under q and pairwise independent. To guarantee that these events can occur in any order from q reaching the same state in any case, we define:

A *weak asynchronous transition system* (wATS) satisfies for all independent events a and b and states q: if $q \stackrel{a}{\rightarrow}$ and $q \stackrel{b}{\rightarrow}$, then there exists some q' with $q \stackrel{ab}{\longrightarrow} q'$ and $q \stackrel{ba}{\longrightarrow} q'$. (This 'forward-diamond-property' only is e.g. also required in [DS93].) For an *asynchronous transition system* (ATS) also $q \stackrel{ab}{\longrightarrow} q'$ implies $q \stackrel{ba}{\longrightarrow} q'$. For convenience, we assume that for each event a of a wATS or ATS there exists some q with $q \stackrel{a}{\rightarrow}$. For an ATS, we furthermore assume that aIb implies that there exists some q with $q \stackrel{a}{\rightarrow}$ and $q \stackrel{b}{\rightarrow}$.

We call wATS A and A' *isomorphic* if one is obtained from the other by a bijective renaming of states that preserves initial states and transition relations.

For comparison of wATSs, we first define ordinary bisimulation: two wATSs A and A' with the same visible events are *(weakly) bisimilar*, if there exists a *bisimulation* between them, i.e. a relation $\mathcal{B} \subseteq Q \times Q'$ such that:

1. $\mathcal{B}$ relates the initial states.
2. If $a \in E_v$, $c \in E_i$ and $q\mathcal{B}q'$, then we have:
   (a) $q \stackrel{a}{\rightarrow} q_1$ implies that for some q_1' $q' \stackrel{a}{\Rightarrow} q_1'$ and $q_1\mathcal{B}q_1'$
   (b) $q \stackrel{c}{\rightarrow} q_1$ implies that for some q_1' $q' \Rightarrow q_1'$ and $q_1\mathcal{B}q_1'$
3. vice versa

A bisimulation *on* A is one between A and A, and states of A (or similarly reachable markings of a net) are *bisimilar* if they are related by a bisimulation on A. For wATS, we also define ST-bisimulations and related notions:

- An *ST-state* (q, μ) of a wATS A consists of a state q and some q-step $\mu \subseteq E_v$. The *initial ST-state* is $(q_0, \emptyset)$.
- For a visible event a, we write $(q, \mu) \stackrel{a^+}{\rightarrow} (q, \mu \cup \{a\})$ if $q \stackrel{a}{\rightarrow}$ and a is independent to all $b \in \mu$ – which again implies $a \notin \mu$. Note that in the case of wATS, we cannot change the state component in a way that would somehow reflect just the starting of a; therefore, we have adapted the notation for nets accordingly in the previous section.

  We write $(q, \mu) \stackrel{a^-}{\rightarrow} (q', \mu \setminus \{a\})$ if $a \in \mu$ and $q \stackrel{a}{\rightarrow} q'$. Finally, for an internal event c, we write $(q, \mu) \stackrel{c}{\rightarrow} (q', \mu)$ if c is independent to all $b \in \mu$ and $q \stackrel{c}{\rightarrow} q'$. Note that in all three cases the pair reached is an ST-state again.
- We again extend this definition to sequences and, by suppressing internal events, to visible sequences.

Two wATSs A, A' with the same visible events are *ST-bisimilar*, if there is an *ST-bisimulation* between them, i.e. a relation $\mathcal{B}$ between their ST-states with:

1. $\mathcal{B}$ relates the initial ST-states.
2. If $a \in E_v$, $c \in E_i$ and $(q,\mu)\mathcal{B}(q',\mu)$ (with the same μ), then we have:
   (a) $(q,\mu) \stackrel{a^+}{\rightarrow} (q,\mu\cup\{a\})$ implies that for some q'' $(q',\mu) \stackrel{a^+}{\Rightarrow} (q'',\mu\cup\{a\})$ and $(q,\mu\cup\{a\})\mathcal{B}(q'',\mu\cup\{a\})$
   (b) $(q,\mu) \stackrel{a^-}{\rightarrow} (q_1,\mu\setminus\{a\})$ implies that for some q_1' $(q',\mu) \stackrel{a^-}{\Rightarrow} (q_1',\mu\setminus\{a\})$ and $(q_1,\mu\setminus\{a\})\mathcal{B}(q_1',\mu\setminus\{a\})$
   (c) $(q,\mu) \stackrel{c}{\rightarrow} (q_1,\mu)$ implies: for some q_1' $(q',\mu) \Rightarrow (q_1',\mu)$ and $(q_1,\mu)\mathcal{B}(q_1',\mu)$
3. vice versa

In this paper, we define the *reachability graph* of a net N to be an ATS: its states are the reachable markings, it has the same visible and internal events as N, the transition relation is given by the firing rule, M_N is the initial state and the independence relation is $I(N)$ restricted to those (a,b) that are enabled under a common reachable marking. Thus, the above definition of ST-bisimulation extends the one from the last section, since nets are ST-bisimilar if and only if their reachability graphs are ST-bisimilar. In this sense, we can also speak of a net being ST-bisimilar to an ATS or wATS.

Similarly, we call two nets *bismilar* if their reachability graphs are bismilar, and this way we can also speak of a net being bisimilar to an ATS or wATS.

History-preserving bisimulations, or hp-bisimulations for short, are usually defined for nets based on the partial orders induced by net processes. These partial orders can alternatively be obtained as Mazurkiewicz traces. Since the latter can be naturally defined for ATS as well (but not for wATS!), we define hp-bisimulation for ATS in this way; via the reachability graph, this also defines when two nets or a net and an ATS are *hp-bisimilar*.

For an ATS A and event sequences v and w, we write $v \sim w$ if, for independent events a and b, $v = uabu'$ and $w = ubau'$. If v and w are related by the reflexive-transitive closure of $\sim$, we call them *equivalent*, write $[v]$ for the equivalence class of v and call it a *trace* – and a *trace of* A, if v is an occurrence sequence of A. Due to the stronger independence requirement for ATS, all elements of a trace of A are occurrence sequences reaching the same state.

To each trace $[a_1 \dots a_n]$ we can associate a labelled partial order on $\{1,\dots,n\}$: each i is labelled with a_i and the order is the least transitive relation where i is 'less than' j if $i < j$ and a_i and a_j are dependent. $[a_1 \dots a_n]$ is exactly the set of linearizations of this labelled partial order, which is up to isomorphism independent of the representative $a_1 \dots a_n$. (Hence, strictly speaking, we consider labelled partial orders only up to isomorphism.) If we restrict the labelled partial order to the visible events, we obtain the *visible po* of $[a_1 \dots a_n]$.

Two ATSs A and A' with the same visible events are *hp-bisimilar*, if there is an *hp-bisimulation* between them, i.e. a relation $\mathcal{B}$ between their traces with:

1. $[\lambda]\mathcal{B}[\lambda]$.
2. If $[v]\mathcal{B}[w]$, then $[v]$ and $[w]$ have the same visible po (up to isomorphism).
3. If $[v]\mathcal{B}[w]$ and $[va]$ is a trace of A for some event a, then there exists some trace $[wu]$, $u \in E'^*$, with $[va]\mathcal{B}[wu]$.
4. vice versa

Again, for general labelled nets, the elements of an hp-bisimulation are in fact triples where the additional component is an explicit isomorphism between the visible partial orders; again this is not necessary here where elements with the same label are always ordered such that the required isomorphism is unique.

4 ST-Bisimilar Implementations of Weak ATSs

In this section, we will construct a net N that is ST-bisimilar to a given visible weak ATS $A = (Q, E_v, \emptyset, T, q_0, I)$. As explained, this can be seen as an improvement on known constructions that avoid equally labelled transitions, since N will be bisimilar to A *and* divergence-free; on top of this, we also consider independence of events.

First of all, we find a family of cliques covering the dependence graph of I, i.e. a family (D_i) of nonempty subsets of E_v such that the elements of each D_i are pairwise dependent and such that for any dependent events a and b there exists a D_i containing a and b; in particular, the union of the D_i is E_v since possibly $a = b$. If one is not interested in concurrency, one can choose $I = \emptyset$ and E_v as the only D_i; Figure 1 shows a transition system and part of our net construction for this simple case, which yields a sequential net.

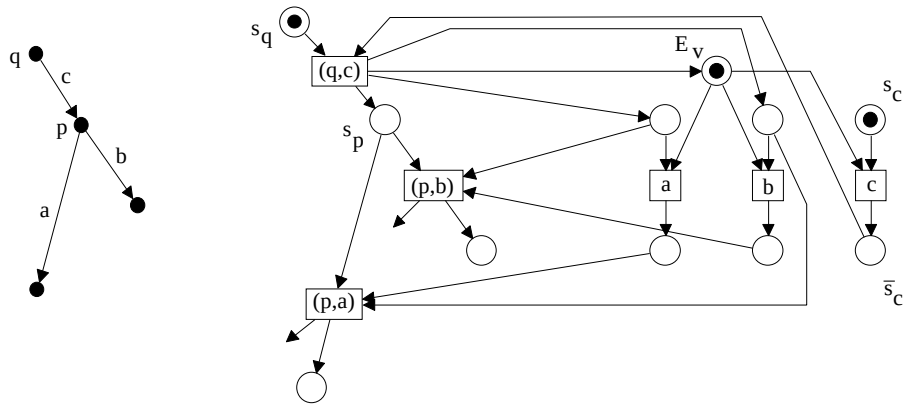


Figure 1

N has places s_q for $q \in Q$, s_a and $\bar{s}_a$ for $a \in E_v$, and D_i. It has the visible events in E_v, and $E_i = \{(q,a) \mid q \in Q,\ a \in E_v,\ q \xrightarrow{a}\}$. We define F by giving the pre- and postsets of the events. For $a \in E_v$, ${}^\bullet a = \{s_a\} \cup \{D_i \mid a \in D_i\}$ and $a^\bullet = \{\bar{s}_a\}$. For an internal event (q,a) with $q \xrightarrow{a} p$, we define:

$$
\begin{aligned}
{}^\bullet(q,a) &= \{s_q\} \cup \{\bar{s}_a\} \cup \{s_b \mid a \neq b \ \wedge\ q \xrightarrow{b} \ \wedge \neg p \xrightarrow{b}\} \\
&\quad \cup \{D_i \mid (\exists b \in D_i :\ q \xrightarrow{b}) \ \wedge\ (\neg\exists b \in D_i :\ p \xrightarrow{b}) \ \wedge\ a \notin D_i\} \\
(q,a)^\bullet &= \{s_p\} \cup \{s_b \mid p \xrightarrow{b} \ \wedge (a = b \vee \neg q \xrightarrow{b})\} \\
&\quad \cup \{D_i \mid (\exists b \in D_i :\ p \xrightarrow{b}) \ \wedge\ (a \in D_i \vee \neg\exists b \in D_i :\ q \xrightarrow{b})\}
\end{aligned}
$$

It will turn out that the reachable markings have the form $M(q,\nu)$ where $\nu \subseteq E_v$ is a visible q-step; we define:

$$M(q,\nu) = \{s_q\} \cup \{s_b \mid q \xrightarrow{b} \ \wedge\ b \notin \nu\} \cup \{\bar{s}_a \mid a \in \nu\}$$
$$\cup\ \{D_i \mid (\exists b \in D_i :\ q \xrightarrow{b}) \ \wedge\ D_i \cap \nu = \emptyset\}$$

With this definition, the initial marking of N is $M(q_0,\emptyset)$.

Lemma 1. *N is safe and divergence-free, and the reachable markings of N are the markings $M(q,\nu)$ where $\nu \subseteq E_v$ is a visible q-step. More in detail:*

1. *$M(q,\nu)$ enables $a \in E_v$ iff $q \xrightarrow{a}$ and a is independent of each event in ν; then, $M(q,\nu)[a\rangle M(q,\nu \cup \{a\})$.*
2. *$M(q,\nu)$ enables an internal event iff it has the form (q,a) with $a \in \nu$ and $q \xrightarrow{a} p$ for some p; then, $M(q,\nu)[(q,a)\rangle M(p,\nu \setminus \{a\})$.*

Proof. First note that divergence-freeness will follow from statement 2. We will consider some $M(q,\nu)$ and show that firing any enabled transition does not violate safety and reaches again a marking of the desired form. Since the initial marking is $M(q_0,\emptyset)$, this shows that N is safe and the reachable markings of N are of the desired form. From our considerations, it will be easy to see that any $M(q,\nu)$ can be reached in N by taking a sequence reaching q in A, inserting after each a occurring from some p in A the internal transition (p,a) and finally adding the events in ν in some order.

If $a \in E_v$ is enabled under $M(q,\nu)$, then a needs a token from s_a, hence $q \xrightarrow{a}$, and a needs a token from all D_i, where $a \in D_i$; thus, all these D_i have an empty intersection with ν and a is independent of each event in ν by choice of the D_i. Vice versa, each a with $q \xrightarrow{a}$ that is independent of each event in ν is easily seen to be enabled under $M(q,\nu)$. Firing such an a gives $M(q,\nu \cup \{a\})$ and does not violate safety.

An internal event enabled under $M(q,\nu)$ needs a token from s_q and from some $\bar{s}_a$, and thus has the form (q,a) with $a \in \nu$ and $q \xrightarrow{a} p$ for some p. By considering the different types of places separately, we show that each such (q,a) is in fact enabled and that firing it does not violate safety and reaches $M(p,\nu \setminus \{a\})$.

Firing (q,a) as above removes $\bar{s}_a$ and replaces s_q by s_p observing safety.

For the places s_b, $b \in E_v$, observe that $b \in \nu$ implies $a = b \vee p \xrightarrow{b}$. Now, $s_b \in {}^\bullet(q,a)$ implies $a \neq b \wedge \neg p \xrightarrow{b}$, hence $b \notin \nu$; since also $q \xrightarrow{b}$, we see that $M(q,\nu)$ marks $s_b \in {}^\bullet(q,a)$ and (q,a) is enabled w.r.t. the s_b. If $M(q,\nu)$ would mark some $s_b \in (q,a)^\bullet$, then $q \xrightarrow{b}$ and thus $b = a \in \nu$, a contradiction. Hence, firing (q,a) does not violate safety w.r.t. the s_b and s_b is marked after firing (q,a) iff $s_b \in (q,a)^\bullet$ or $s_b \notin {}^\bullet(q,a) \wedge s_b \in M(q,\nu)$. The latter disjunct means $(a = b \vee \neg q \xrightarrow{b} \vee\ p \xrightarrow{b}) \wedge q \xrightarrow{b} \wedge\ b \notin \nu$, i.e. $p \xrightarrow{b} \wedge\ q \xrightarrow{b} \wedge\ b \notin \nu$, since $a = b$ contradicts $b \notin \nu$. Expanding $s_b \in (q,a)^\bullet$, we get that s_b is marked after firing (q,a) iff $p \xrightarrow{b}$ and $a = b \vee \neg q \xrightarrow{b} \vee\ (q \xrightarrow{b} \wedge\ b \notin \nu)$. Since $b \in \nu$ implies $q \xrightarrow{b}$, the latter conjunct means $a = b \vee b \notin \nu$, i.e. $b \notin \nu \setminus \{a\}$. Thus, s_b is marked after firing (q,a) iff $p \xrightarrow{b}$ and $b \notin \nu \setminus \{a\}$ iff $M(p,\nu \setminus \{a\})$ marks s_b.

It remains to check the D_i, so let us fix one of them; we will write $A(q)$ for $\exists b \in D_i : q \xrightarrow{b}$ and analogously $A(p)$. Furthermore, B will stand for $a \in D_i$ and C for $D_i \cap (\nu \setminus \{a\}) = \emptyset$. With this, we have

$D_i \in {}^\bullet(q,a)$ iff $A(q) \wedge \neg A(p) \wedge \neg B$,
$D_i \in (q,a)^\bullet$ iff $A(p) \wedge (B \vee \neg A(q))$,
$D_i \in M(q,\nu)$ iff $A(q) \wedge \neg B \wedge C$,
$D_i \in M(p, \nu \setminus \{a\})$ iff $A(p) \wedge C$.

We list a number of properties:

i) $\neg A(p)$ implies C. If $\neg C$, pick some $b \in D_i \cap (\nu \setminus \{a\})$; then aIb and $q \xrightarrow{b}$, thus $q \xrightarrow{a} p$ implies $p \xrightarrow{b}$ and, hence, $A(p)$.

ii) $D_i \in {}^\bullet(q,a)$ implies $D_i \in M(q,\nu)$ by i); thus, (q,a) is enabled w.r.t. the D_i.

iii) $D_i \in M(q,\nu)$ implies $D_i \notin (q,a)^\bullet$ (because of $A(q) \wedge \neg B$). Hence, firing (q,a) does not violate safety w.r.t. the D_i.

iv) B implies C. (a is independent to all other events in ν.)

v) $\neg A(q)$ implies C, since $b \in D_i \cap (\nu \setminus \{a\})$ implies $b \in \nu$, thus $q \xrightarrow{b}$.

Now D_i is marked after firing (q,a) iff $D_i \in (q,a)^\bullet$ or $D_i \notin {}^\bullet(q,a) \wedge D_i \in M(q,\nu)$. The latter disjunct means $(\neg A(q) \vee A(p) \vee B) \wedge A(q) \wedge \neg B \wedge C$, which can be simplified to $A(p) \wedge A(q) \wedge \neg B \wedge C$. Expanding $D_i \in (q,a)^\bullet$, we get that D_i is marked after firing (q,a) iff $A(p)$ and $B \vee \neg A(q) \vee (A(q) \wedge \neg B \wedge C)$. The latter conjunct can be simplified to $B \vee \neg A(q) \vee C$ and by iv) and v) to C. Thus, D_i is marked after firing (q,a) iff $A(p)$ and C iff $M(p, \nu \setminus \{a\})$ marks D_i. □

Now we will show that $\mathcal{B}$ is an ST-bisimulation between A and N, where $\mathcal{B}$ relates (q',μ) to $(M(q,\nu),\mu)$ if $\mu \cup \nu$ is a visible q-step, $\mu \cap \nu = \emptyset$ and from q the step ν reaches q' (which then allows the step μ). This is clearly satisfied for $(q_0,\emptyset)$ and $(M(q_0,\emptyset),\emptyset)$, so let us assume $(q',\mu)\mathcal{B}(M(q,\nu),\mu)$.

We first show how A simulates N, so consider $(M(q,\nu),\mu)[a^+\rangle(M(q,\nu),\mu \cup \{a\})$, hence a is enabled under $M(q,\nu)$ and independent to all events in μ. a is also independent to all events in ν, since otherwise some $D_i \in {}^\bullet a$ would be missing in $M(q,\nu)$ – in particular, $a \notin \nu$. Since s_a must be marked under $M(q,\nu)$, we get $q \xrightarrow{a}$; thus, $\mu \cup \nu \cup \{a\}$ is a visible q-step and $\mu \cup \{a\}$ is a visible q'-step; in particular, $(q',\mu) \xrightarrow{a^+} (q',\mu \cup \{a\})$. Furthermore, $(q',\mu \cup \{a\})\mathcal{B}(M(q,\nu),\mu \cup \{a\})$.

Next, consider $(M(q,\nu),\mu)[a^-\rangle(M(q,\nu \cup \{a\}),\mu \setminus \{a\})$ – see Lemma 1. Since q' enables $a \in \mu$, take p with $q' \xrightarrow{a} p$, i.e. $(q',\mu) \xrightarrow{a^-} (p,\mu \setminus \{a\})$. Now $(\nu \cup \{a\}) \cup (\mu \setminus \{a\}) = \mu \cup \nu$ is a visible q-step, from q the step $\nu \cup \{a\}$ reaches p and $(\nu \cup \{a\}) \cap (\mu \setminus \{a\}) = \emptyset$; thus, $(p,\mu \setminus \{a\})\mathcal{B}(M(q,\nu \cup \{a\}),\mu \setminus \{a\})$.

To conclude with this simulation, consider $(M(q,\nu),\mu)[(q,a)\rangle(M(p,\nu\setminus\{a\}),\mu)$ where $a \in \nu$ and $q \xrightarrow{a} p$ – see Lemma 1. Since $(q',\mu)\mathcal{B}(M(q,\nu),\mu)$, $\mu \cup \nu \setminus \{a\}$ is a visible p-step, where $\mu \cap (\nu \setminus \{a\}) \subseteq \mu \cap \nu = \emptyset$, and from p the step $\nu \setminus \{a\}$ reaches q'; thus, $(q',\mu)\mathcal{B}(M(p,\nu \setminus \{a\}),\mu)$.

Now, we show how N simulates A; given $(q',\mu)\mathcal{B}(M(q,\nu),\mu)$, we can conclude from the last paragraph that N can first fire internal events reaching

$(M(q',\emptyset),\mu)$, which is $\mathcal{B}$-related to (q',μ) as well. For this, we have to show that (q,a) with $a \in \nu$ and $q \xrightarrow{a} p$ for some p is independent of the events in μ, so take some $b \in \mu$, i.e. $a \neq b$. The places in ${}^\bullet b \cup b^\bullet$ are $\bar{s}_b$, which is not in ${}^\bullet(q,a) \cup (q,a)^\bullet$, s_b and each D_i with $b \in D_i$. Since $\mu \cup \nu$ is a q-step, we have $q \xrightarrow{b}$ and $p \xrightarrow{b}$; thus, s_b is not in ${}^\bullet(q,a) \cup (q,a)^\bullet$, either. Since $p \xrightarrow{b}$, a D_i containing b is not in ${}^\bullet(q,a)$. If such a D_i were in $(q,a)^\bullet$, $q \xrightarrow{b}$ would imply $a \in D_i$, but a and b are independent. Thus, we have shown the independence of (q,a) and b.

Hence, we only have to consider the case $(q',\mu)\mathcal{B}(M(q',\emptyset),\mu)$. On the one hand, if $(q',\mu) \xrightarrow{a^+} (q',\mu \cup \{a\})$, then $a \notin \mu$ and $\mu \cup \{a\}$ is a q'-step, i.e. $q' \xrightarrow{a}$ and a is independent of the events in μ. With Lemma 1 we get that $M(q',\emptyset)$ enables a, thus $(M(q',\emptyset),\mu)[a^+\rangle(M(q',\emptyset),\mu \cup \{a\})$ and the latter is $\mathcal{B}$-related to $(q',\mu \cup \{a\})$.

On the other hand, if $(q',\mu) \xrightarrow{a^-} (p,\mu \setminus \{a\})$ with $a \in \mu$ and $q' \xrightarrow{a} p$, we have $M(q',\emptyset)[a\rangle M(q',\{a\})$ by Lemma 1. Hence, $(M(q',\emptyset),\mu)[a^-\rangle(M(q',\{a\}),\mu \setminus \{a\})$ and the latter ST-marking is $\mathcal{B}$-related to $(p,\mu \setminus \{a\})$. Thus we have shown:

Theorem 2. *For each visible weak ATS A there exists a divergence-free net N that is ST-bisimilar to A.*

Note that N constructed from A above contains a loop if and only if A contains a loop. Such a loop in N consists of (q,a) and s_q with $q \xrightarrow{a} q$.

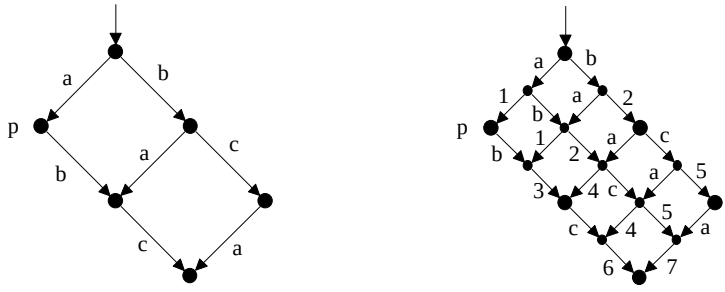


Figure 2

Figure 2 shows an ATS, where a is independent of b and c, and the reachability graph of the net that results from our construction, where the internal events have simply been numbered and $1 = (q_0,a)$, $2 = (q_0,b)$ and $3 = (p,b)$. This example demonstrates the transformation of the given ATS that is implied by our construction; one could call it τ-splitting of events where τ stands for an internal event. But it also makes clear that this construction does not always give a 'good' result: the occurrence sequence $ab13c$ gives a trace where, in the visible po, c comes 'after', i.e. depends causally on, both a and b. Of course, it is straightforward to find a net N with the given ATS as reachability graph, such that in N c is completely independent of a.

But note that our construction works for any *weak* asynchronous transition system; Figure 3 shows a transition system and a net implementation (not ob-

tained by our construction) where a and b are independent and state q satisfies $q \xrightarrow{ab}$ but not $q \xrightarrow{ba}$. This is only possible when using internal events in the net.

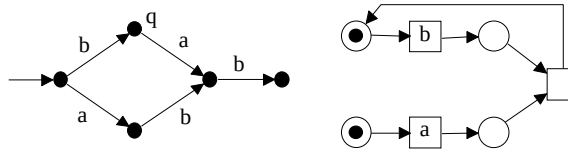


Figure 3

5 Safe and Semi-safe Transition Systems

For Section 6, we need the classical theory of regions as extended to safe nets in [PKY95,DS93] and as varied in [CKLY98]. To allow reference to the proofs there we present this material, also allowing loops in transition systems in contrast to [CKLY98]. In this section, all events are visible, i.e. the additional feature of internal events is of no importance and we are in the more usual setting. Also, our results do not depend on independence; independence is just an additional feature that makes Theorem 6 below stronger using Proposition 5.

For a visible ATS A, a *region* is a nonempty, proper subset R of Q satisfying for each event a one of the following cases (where the third is a subcase of the fourth):

- R is a *pre-region* of a, $R \in {}^\circ a$, i.e. $q \xrightarrow{a} p$ implies $q \in R$ and $p \notin R$.
- R is a *post-region* of a, $R \in a^\circ$, i.e. $q \xrightarrow{a} p$ implies $q \notin R$ and $p \in R$.
- R is a *co-region* of a, $R \in \overset{\circ}{a}$, i.e. $q \xrightarrow{a} p$ implies $q \in R$ and $p \in R$.
- a is *not crossing* R, i.e. $q \xrightarrow{a} p$ implies $q, p \in R$ or $q, p \notin R$.

A visible ATS is *safe*, if it satisfies the following two properties:

- *event separation*: for all $a \in E$ and $q \in Q$, $\neg q \xrightarrow{a}$ implies that there exists a region $R \in {}^\circ a \cup \overset{\circ}{a}$ with $q \notin R$.
- *state separation*: for all $p, q \in Q$ with $p \neq q$, there exists a region R such that $p \in R$ iff $q \notin R$.

We will show that safe ATSs are (up to isomorphism) just the reachability graphs of general safe nets, where one implication is quite obvious.

Proposition 3. *If N is a visible net, then its reachability graph is a safe ATS.*

For the other implication, we first note:

Lemma 4. *If A is a safe ATS and a an event that is enabled under all states, then $q \xrightarrow{a} q$ for all states q.*

We call an event a with $q \xrightarrow{a} q$ for all states q a *loop-event*.

Now assume we are given a safe ATS A; let $\mathcal{R}$ be a set of regions that are sufficient to satisfy event and state separation. We construct a net $N = (\mathcal{R} \cup \{s\}, E_v, \emptyset, F, M_N)$ by letting $(R,a) \in F$ if $R \in {}^\circ a \cup \overset{\circ}{a}$, $(a,R) \in F$ if $R \in a^\circ \cup \overset{\circ}{a}$, $(a,s),(s,a) \in F$ if a is a loop-event, and finally $M_N = \{R \in \mathcal{R} \mid q_0 \in R\} \cup \{s\}$. First note, that each event a has a nonempty preset in N since it is either a loop-event or it has a region R with $(R,a) \in F$ by event separation.

We now show that the reachability graph of N is isomorphic to A except for the independence relation, where $q \in Q$ corresponds to $\{R \in \mathcal{R} \mid q \in R\} \cup \{s\}$, which is injective by state separation of $\mathcal{R}$ and obviously preserves initial states.

We will show that this correspondence preserves the transition relation, too, and also check that safety is not violated in N. Since all states are reachable, this shows at the same time that our correspondence is a bijection onto the reachable markings. Observe that a loop-event has only s in its pre- and postset, which is always marked; thus, we can ignore s and any loop-events in the following. Now, take corresponding q and M and first assume $q \xrightarrow{a} q'$. If $(R,a) \in F$, then $q \in R$ and R is marked under M; thus, a is enabled under M; we define $M' = (M \setminus {}^\bullet a) \cup a^\bullet$. We check the different possibilities for a region $R \in \mathcal{R}$ w.r.t. a. If $R \in {}^\circ a$, then on the one hand $q \in R$ and $q' \notin R$, while on the other hand firing a empties R; hence $R \notin M'$. If $R \in a^\circ$, then on the one hand $q \notin R$ and $q' \in R$, while on the other hand firing a marks R; hence $R \notin M$, such that safety is not violated, and $R \in M'$. If $R \in \overset{\circ}{a}$, then on the one hand $q \in R$ and $q' \in R$, while on the other hand a and R form a loop; hence safety is not violated and $R \in M'$. If none of these cases applies, then on the one hand $q \in R$ iff $q' \in R$, while on the other hand firing a does not change the marking of R; hence, $R \in M$ iff $R \in M'$. In any case, q' and M' correspond (using the correspondence of q and M in the last case).

Second, assume $M[a\rangle M'$. If $\neg q \xrightarrow{a}$, then there is some $R \in \mathcal{R}$ with $q \notin R$ and $R \in {}^\circ a \cup \overset{\circ}{a}$; this implies $R \notin M$ and $(R,a) \in F$, a contradiction. Hence, $q \xrightarrow{a} q'$ for some q', which by the last paragraph corresponds to M'.

If we are not interested in the independence relation, N realizes A. Otherwise, note that our correspondence is an isomorphism (up to the independence relation) and hence also a bisimulation, and apply the following new result.

Proposition 5. *Let A be a visible ATS and N a visible net bisimilar to A. Then there exists a net N' whose reachability graph is isomorphic to that of N except that its independence is I (i.e. that of A).*

Proof. If we have events a and b that are independent according to $I(N)$ but not according to I, we can add a common marked loop place to a and b. This does not really change the reachability graph of N and makes a and b dependent.

We also could have events a and b that are independent according to I but not according to $I(N)$. Consider some $q_1 \in Q$ with $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_4$ and $q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_4$. Due to bisimilarity, there is a reachable marking M_1 of N with $M_1[a\rangle M_2[b\rangle M_4$ and $M_1[b\rangle M_3[a\rangle M_4'$. The effect of firing a and b does not depend on their order;

hence $M_4 = M_4'$. Furthermore, a place in $({}^\bullet a \cup a^\bullet) \cap ({}^\bullet b \cup b^\bullet)$ can only be a common loop place of a and b. In this situation, we duplicate each such common loop place s such that both copies are connected by arcs to all other events in the same way as s, but one copy is on a loop with a (and not with b) while the other is on a loop with b (and not with a). This does not really change the reachability graph of N and makes a and b independent. □

We conclude:

Theorem 6. *If A is a safe ATS, then there effectively exists a (visible) net N whose reachability graph is isomorphic to A.*

As a next step, we weaken the definition of a safe ATS. A visible ATS is *semi-safe* if it satisfies event separation, but not necessarily state separation. Thus, a semi-safe ATS is up to the treatment of loops and up to independence what [CKLY98] calls an excitation-closed transition system. Following [CKLY98], we will show that a semi-safe ATS A can be realized by a visible net up to bisimilarity by transforming A to a bisimilar safe ATS A' and applying the above theorem. Semi-safety on languages, i.e. on infinite tree-shaped transition systems is also considered in [BBD95] and in the context of trace languages in [HKT92]. We note a lemma first.

Lemma 7. *Let A be a visible ATS, $\mathcal{B}$ an equivalence on Q that is also a bisimulation on A, and denote the equivalence class of $q \in Q$ by $[q]$. Then the $\mathcal{B}$-quotient $A' = (\{[q] \mid q \in Q\}, E_v, \emptyset, T', [q_0], I)$ is a (visible) ATS bisimilar to A, where $T'([q], a) = [q']$ if $q \xrightarrow{a} q'$.*

Now assume a semi-safe ATS A is given. Define a relation $\mathcal{B}$ on Q by $q\mathcal{B}p$ if q and p are contained in the same regions. Clearly, $\mathcal{B}$ is an equivalence.

Let $q \xrightarrow{a} q'$ and $q\mathcal{B}p$; if $\neg p \xrightarrow{a}$, then due to event separation there would be a pre- or co-region of a – hence containing q – not containing p, a contradiction. Thus, there is some p' with $p \xrightarrow{a} p'$. If a region R contains q' and q, hence p, then a is not crossing R, i.e. $p' \in R$. If R contains q' but neither q nor p, then R is a post-region of a and contains p'. Vice versa, each region containing p' contains q', too; thus, $q'\mathcal{B}p'$. Therefore, $\mathcal{B}$ is a bisimulation on A. By the above lemma, A' as defined there is an ATS bisimilar to A.

It remains to show that in our case A' is a safe ATS. For a region R of A, we define $[R] = \{[q] \mid q \in R\}$. If R is a region of A and $q \in R$, then clearly $[q] \subseteq R$. With this and the above considerations, it is not hard to see that $[R]$ is a region of A' and, more precisely, a pre-, co- or post-region for some a if R is a pre-, co- or post-region for this a in A. Thus, if some $[q]$ does not enable some a in A', then neither does q in A and there is some pre- or co-region R of a not containing q; hence, $[R]$ is a pre- or co-region of a not containing $[q]$. If $[q] \neq [p]$, then $\neg q\mathcal{B}p$ and there is some region R containing exactly one of p and q, thus $[R]$ is a region containing exactly one of $[p]$ and $[q]$. We conclude:

Theorem 8. *If A is a semi-safe ATS, then there effectively exists a bisimilar safe ATS and a visible net N bisimilar to A.*

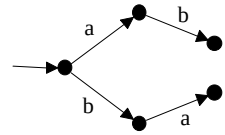


Figure 4

Figure 4 shows a semi-safe ATS (with dependent a and b). Clearly, it cannot be the reachability graph of a net, but identification of the two 'terminal' states gives such an ATS.

6 Results on Bisimilar and hp-Bisimilar Implementations of ATSs

Largely repeating from the literature, we have seen in the last section that semi-safe ATSs can be implemented as visible nets up to bisimilarity. We started out with the aim to use behaviour notions that also consider concurrency. With the following easy lemma, it becomes obvious from Proposition 5 that, whenever we can realize a visible ATS by a bisimilar visible net, we can also realize it by a hp-bisimilar visible net; this applies in particular to semi-safe ATSs.

Lemma 9. *If two bisimilar visible ATSs have the same independence relation, then they are hp-bisimilar.*

Proof. By bisimilarity, the two visible ATSs have the same occurrence sequences, hence the same traces, and the identity is an hp-bisimulation. □

Corollary 10. *Let A be a visible ATS and N a visible net bisimilar to A. Then there also exists a visible net hp-bisimilar to A. In particular, if A is a semi-safe ATS, then there exists a visible net hp-bisimilar to A.*

While we have seen in the last section that identifying bisimilar states in an ATS can help to find a net implementation, Figure 5 shows an ATS that violates event separation for d and q; since there are no bisimilar states, identification of bisimilar states cannot help here. Nevertheless, the ATS has a bisimilar net implementation as shown where the two occurrences of c lead to different markings. So splitting a state into bisimilar copies can also help.

Semi-safety is sufficient to ensure that a bisimilar visible net exists; we will now give an algorithm to *decide* whether to a given visible ATS there exists a bisimilar (or hp-bisimilar) visible net. We list some lemmata first.

Lemma 11. *Let A' and A be ATSs with the same events and h a morphism from A' to A, i.e. a function from Q' to Q with $h(q'_0) = q_0$ such that $p \xrightarrow{a} q$ in A' implies $h(p) \xrightarrow{a} h(q)$ in A. If R is a region of A (and a pre-/post-/co-region of some a), then $h^{-1}(R)$ is a region of A' (and a pre-/post-/co-region of a).*

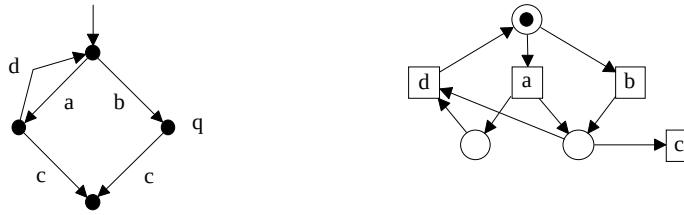


Figure 5

Lemma 12. *Let A and A' be bisimilar visible ATSs and define $q\mathcal{B}q'$ if $q_0 \xrightarrow{w} q$ in A and $q'_0 \xrightarrow{w} q'$ in A' for some $w \in E^*$. Then $\mathcal{B}$ is a bisimulation.*

Lemma 13. *Let N be a visible net and M, M' be reachable markings with $M[w\rangle M'$ for some $w \in E^*$. If $M'[w\rangle$ or M and M' are bisimilar, then $M = M'$.*

Let A be an ATS. States p and q of A are *strongly connected*, if there are paths from p to q and from q to p (i.e. $p \xrightarrow{v} q$ and $q \xrightarrow{w} p$ for some $v, w \in E^*$). Being strongly connected is an equivalence relation; a *strongly connected component* (*scc* for short) consists of an equivalence class together with the arcs between any two of its elements. An arc $p \xrightarrow{a} q$ is a *tree-edge*, if it does not belong to any *scc*. Note that no path in A can use a tree-edge twice, because then we would have a cycle containing the tree-edge, and this cycle would be contained in a *scc*.

We will now define the *scc-tree* of A, denoted *scc-tree*(A). The idea is to unfold A into a tree-like ATS, where the *scc*s are left intact but possibly get duplicated, and form a tree with the tree-edges. This unfolding gives a finite ATS (in contrast with the complete unfolding into a tree), but all states of A are split as much as it could be helpful for finding a suitable net.

Let $q_0 \xrightarrow{a_1} q_1$, $q_1 \xrightarrow{a_2} q_2$, ..., $q_{n-1} \xrightarrow{a_n} q_n$ in A; taking the subsequence of tree-edges and representing each such tree-edge $q \xrightarrow{a} p$ as (q, a, p), we obtain a sequence σ which we call a *tree-path to* q_n. Note that σ cannot contain a repetition, and thus there can only be finitely many tree-paths to any q.

We define *scc-tree*(A) $=: A'$ as follows: $Q' = \{(q, \sigma) \mid \sigma \text{ a tree-path to } q \in Q\}$, $E'_v = E_v$, $E'_i = E_i$, $q'_0 = (q_0, \lambda)$ and $I' = \emptyset$; $T'((q, \sigma), a)$ is (p, σ) if $q \xrightarrow{a} p$ and q and p belong to the same *scc*, and it is $(p, \sigma(q, a, p))$ if $q \xrightarrow{a} p$ is a tree-edge. Observe: both images are indeed in Q', and the transition relation is deterministic.

Occurrence of an event changes the first component of a state of A' as in A, while the second gives just additional information splitting q into several copies. Hence, – just as in A – each event is enabled under some state of A' and A and A' are bisimilar; since the independence relation is empty, A' is an ATS.

Lemma 14. *scc-tree(A) is an ATS and bisimilar to A.*

The following theorem shows that by constructing *scc-tree*(A) we have performed all state splittings that can possibly help to implement A.

Theorem 15. *There exists a visible net N (hp-)bisimilar to a visible ATS A if and only if scc-tree(A) is a semi-safe ATS.*

Proof. Theorem 8, Lemma 14 (and Corollary 10) give the reverse implication.

For the other implication, *scc-tree*(A) and N are bisimilar by Lemma 14, and the relation $\mathcal{B}$ in Lemma 12 relates each state (q,σ) to some reachable marking M; we show that this M is unique, i.e. $\mathcal{B}$ is a function.

If $\sigma = (q_1,a_1,q_1')\dots(q_k,a_k,q_k')$, then each occurrence sequence to (q,σ) uses each arc from $(q_i,(q_1,a_1,q_1')\dots(q_{i-1},a_{i-1},q_{i-1}'))$ to $(q_i',(q_1,a_1,q_1')\dots(q_i,a_i,q_i'))$ for $i=1,\dots,k$. If (q,σ) is related to M and M' due to some w and w', then both of the latter must use these arcs. Thus, we can apply induction if we show:

$(*)$ Assume that $(q_0,\lambda) \xrightarrow{w} (q,\sigma) \xrightarrow{w'} (q',\sigma)$ and $(q,\sigma) \xrightarrow{w''} (q',\sigma)$ in *scc-tree*(A) and that $M_N[w\rangle M[w'\rangle M'$ and $M[w''\rangle M''$ in N; then $M'=M''$.

Proof of $(*)$: Since q and q' belong to the same *scc*, there is some v with $q' \xrightarrow{v} q$, i.e. $(q',\sigma) \xrightarrow{v} (q,\sigma)$. Since $\mathcal{B}$ is a bisimulation, this shows $M[w'\rangle M'[v\rangle M'''$ for some M''' bisimilar to (q,σ), which implies $M'''[w'v\rangle$ and with Lemma 13 $M=M'''$. Thus $M'[vw''\rangle M''$.

Since M' and M'' are related to the same (q',σ) by the bisimulation $\mathcal{B}$, they are bisimilar and hence equal by Lemma 13 again. $\square(*)$

Thus, $\mathcal{B}$ is a function h and in fact a morphism as defined in Lemma 11, which gives us event separation: If $\neg(q,\sigma) \xrightarrow{a}$ in *scc-tree*(A), then $\neg h(q,\sigma)[a\rangle$ in N, since h is a bisimulation. Hence, there is some pre- or co-region R of a in the reachability graph of N with $h(q,\sigma) \notin R$. Thus, $h^{-1}(R) \in {}^\circ a \cup \mathring{a}$ in *scc-tree*(A) with $(q,\sigma) \notin h^{-1}(R)$. We conclude that *scc-tree*(A) is semi-safe. $\square$

Without internal events, finding a bisimilar net is the same problem as finding a language equivalent net. This problem has been considered for unbounded nets in [HKT92] (in a trace setting), for bounded nets (without loops) in [BBD95] and for unbounded nets (without loops) in [Dar98]. It is shown that the language, i.e. the complete unfolding of the transition system, has to satisfy event separation. So the important point of our result is that *scc-tree*(A) is finite. [BBD95,Dar98] give effective results, where [BBD95] works on regular expressions, while [Dar98] uses a finite tree-like unfolding of the transition system; this unfolding seems to be much more complicated than ours – where of course the problem considered is different and unbounded nets are more involved than safe ones.

There can be exponentially many tree-paths in a visible ATS A, e.g. if it consists of a sequence of states each being connected to the next by *two* arcs. On the other hand, if A has n states, there are at most n^n tree-paths, so *scc-tree*(A) can be at most of exponential size compared to A.

To make A and, thus, *scc-tree*(A) smaller, one can first of all replace A by its bisimilarity-quotient A'. Since bisimilarity on A is a bisimulation and an equivalence, A' is a visible ATS bisimilar to A by Lemma 7, so the existence of a bisimilar net can be checked for A' instead of A.

In fact, I assume that in practice the simplicity of the *scc-tree* will help a lot. In many cases, A or at least its bisimilarity-quotient A' will be strongly

connected or consist of a path from the initial state to some *scc* (which is then the only one which may contain more than one state). Then A' is isomorphic to *scc-tree*(A'), and we simply have to check whether A' is semi-safe. Since there are no bisimilar states in A', this is the same as checking safety by the proof of Theorem 8. Of course, semi-safety is most likely easier to check than safety; at least in this case, it is sufficient to find enough regions to guarantee event-separation – also for the construction of a suitable net. We summarize:

Corollary 16. *Let A be a visible ATS and A' be its bisimilarity-quotient.*

i) If A is strongly connected or consists of a path from the initial state to some scc, the same applies to A'. If A' is strongly connected or consists of a path from the initial state to some scc, then scc-tree(A') is isomorphic to A'.

ii) Assume scc-tree(A') is isomorphic to A'. Then there exists a visible net N (hp-)bisimilar to A if and only if A' is safe if and only if A' is semi-safe, where N can directly be constructed from some regions guaranteeing event-separation.

In the cases not covered by this corollary, there are possibilities for further improvements. First, minimizing A possibly merges states that are then split again in *scc-tree*(A); one could try to avoid this, but this would need careful consideration: e.g. just merging bisimilar states that belong to the same *scc* could create non-determinism, hence further merging could be required. Second, if there is some 'diamond' – i.e. $q \xrightarrow{ab} p$ and $q \xrightarrow{ba} p$ – and some arc involved is a tree-edge, then the diamond would be split in *scc-tree*(A); this cannot help to find a net, since in a net firing ab or ba leads to the same marking; thus, splitting in the construction of *scc-tree*(A) could be reduced such that diamonds are kept intact.

We now come to our last contribution: we will exhibit a construction on ATSs that preserves hp-bisimilarity and involves internal events. The example treated in [Yak98] demonstrates that this can turn a visible ATS that is not semi-safe into one that is safe if one regards the additional internal event as visible. This shows how to use our construction: the new ATS is isomorphic to the reachability graph of a net N which is therefore hp-bisimilar to the original ATS if we regard the additional event of N as internal again. (In formal words, this is true because hp-bisimilarity is a congruence for hiding.) Later, we will give an example showing that the construction is not always helpful. Thus, for the time being, we can only offer a trial-and-error method that may help to find a hp-bisimilar net.

A *τ-state-splitting* A' of an ATS A is obtained by choosing a state q and a family $\{q_a \xrightarrow{a} q\}$ of arcs such that a and b are dependent, whenever $q_a \xrightarrow{a} q$ is in this family and we have $p \xrightarrow{b} q$ outside the family or $q \xrightarrow{b} p$ for some state p. Then, A' has a new state q', a new internal event τ dependent to all other events, and a new arc $q' \xrightarrow{\tau} q$; each arc $q_a \xrightarrow{a} q$ of the family is redefined to $q_a \xrightarrow{a} q'$.

Theorem 17. *Let A be an ATS and A' be a τ-state-splitting of A. Then A and A' are hp-bisimilar and one is divergence-free if the other one is.*

Proof. The claim about divergence is easy. We will show the theorem for a visible ATS A; this is sufficient, since turning visible events into internal ones preserves hp-bisimilarity. Assume we are given q and the family of arcs as above.

First, we have to show that A' is an ATS, i.e. satisfies the independence requirements. So assume that in A' $p \xrightarrow{a} p_1$ and for some b independent of a $p \xrightarrow{b}$ or $p_1 \xrightarrow{b}$. Clearly, $p \neq q'$, since τ is not independent of any event. If $p_1 = q'$, we would have $p \xrightarrow{a} q$ and $p \xrightarrow{b}$ in A, hence $q \xrightarrow{b}$ contradicting the choice of the arc family. Thus, we have a diamond $p \xrightarrow{a} p_1 \xrightarrow{b} p_3$ and $p \xrightarrow{b} p_2 \xrightarrow{a} p_3$ in A, and by the above (and symmetry) $p \xrightarrow{a} p_1 \xrightarrow{b}$ and $p \xrightarrow{b} p_2 \xrightarrow{a}$ in A'. If we do not have the same diamond in A', then w.l.o.g. $p_1 \xrightarrow{b} q'$ in A' and this is one of the redefined arcs; by aIb and choice of the arc family, then $p_2 \xrightarrow{a} p_3$ also belongs to the family, i.e. $p_2 \xrightarrow{a} q'$ in A'.

The hp-bisimulation matches each occurrence sequence w (or its trace) in A with the same sequence in A', where we insert a τ after each a arising from some $q_a \xrightarrow{a} q$ in A from the chosen family; clearly, this is an occurrence sequence in A' ending in the same state, and we only have to check that it has the same visible po. So assume that $w = uabu'$ and we have to insert a τ after a. On the A'-side, each visible event before this τ is less than this τ which in turn is less than each visible event after this τ in the full labelled partial order defined from the extended occurrence sequence; we have to check that on the A-side each event in ua is less than each event in bu', too and we will write here $<$ for less than. The a arises from an arc into q, the b from an arc out of q, hence a and b are dependent and $a < b$; so choose some (occurrence of some) c in u and d in u'. If $c < a$, also $c < b$; otherwise, we can commute the c behind a, i.e. there is a c-labelled arc into q and aIc; by choice of the arc family, this arc belongs to it, b and c are dependent and $c < b$. If $b < d$ we are done; otherwise, we find a d-labelled arc from q and by choice of the family $a < d$; if now $\neg c < a$, then we have a c-labelled arc into q belonging to the family and a d-labelled arc from q, thus by choice of the family $c < d$. □

As a negative example, consider simply a diamond with arcs $p \xrightarrow{a} q$ and $q \xrightarrow{b} p'$, but with a and b being dependent. This can easily be realized by a net, but inserting a τ between the two arcs makes it impossible – the τ-transition would have to have a zero-effect, but would also be required to change the marking. Of course, this can be remedied by inserting another internal event on the other side of the diamond.

We conclude by the remark that our construction above coincides with the one in Section 4 in an extreme case, namely if all events are dependent and we apply the former to each arc separately.

7 Concluding Remark

The results in this paper are only a beginning. Clearly, the problem of finding a history-preserving bisimilar implementation for an asynchronous transition system has only been touched upon, although the author does not know of any

asynchronous transition system that does not have such an implementation. Furthermore, for realistic application it is necessary to optimize the implementation by minimizing the number of places (as it is e.g. done in [CKLY98]) or – most of all – the number of additional internal transitions.

References

BBD95. E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In P. Mosses et al., editors, *TAPSOFT 95*, Lect. Notes Comp. Sci. 915, 364–378. Springer, 1995.

BDKP91. E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Informatica*, 28:231–264, 1991.

CKLY98. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47:859–882, 1998.

Dar98. P. Darondeau. Deriving unbounded Petri nets from formal languages. In D. Sangiorgi and R. de Simone, editors, *CONCUR 98*, Lect. Notes Comp. Sci. 1466, 533–548. Springer, 1998.

DS93. M. Droste and R.M. Shortt. Petri nets and automata with concurrency relations – an adjunction. In M. Droste et al., editors, *Semantics of Programming Languages and Model Theory*. Gordon and Breach, 69–87. 1993.

GV87. R.J. v. Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker et al., editors, *PARLE Vol. II*, Lect. Notes Comp. Sci. 259, 224–242. Springer, 1987.

HKT92. P. Hoogers, H. Kleijn, and P.S. Thiagarajan. A trace semantics for Petri nets. In W. Kuich, editor, *Automata, Languages and Programming, ICALP 1992*, Lect. Notes Comp. Sci. 623, 595–604. Springer, 1992.

NRT92. M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theor. Comput. Sci.*, 96:3–33, 1992.

NW95. M. Nielsen and G. Winskel. Models for concurrency. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science Vol. 4*, pages 1–148. Oxford Univ. Press, 1995.

PKY95. M. Pietkiewicz-Koutny and A. Yakovlev. Non-pure nets and their transition systems. Technical Report TR 528, Dept. Comp. Sci., Univ. of Newcastle upon Tyne, 1995.

Sun98. K. Sunesen. *Reasoning about Reactive Systems*. PhD thesis, Univ. Aarhus, 1998.

Vog92. W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*. Lect. Notes Comp. Sci. 625. Springer, 1992.

Vog95. W. Vogler. Timed testing of concurrent systems. *Information and Computation*, 121:149–171, 1995.

Yak98. A. Yakovlev. Designing control logic for counterflow pipeline processor using Petri nets. *Formal Methods in System Design*, 12:39–71, 1998.

Trace Channel Nets

Jean Fanchon

LAAS du CRNS, 7 av. du Col. Roche, 31077 Toulouse, France
fanchon@laas.fr

Abstract. We present a new class of nets which includes and extends both Coloured Nets and Fifo Nets by defining weights on edges and markings of places as traces on a concurrent (trace) alphabet. Considering different independence relations on the alphabet, from the maximal one to the empty one (yielding words), Trace Channel Nets open a hierarchy of semantics on a single net structure. Furthermore a field of investigation results from the relationship between the independence on the alphabet and the behaviours of the net. In particular we show that the boundedness of a TCNet is related to particular independence relations, maximal w.r.t. boundedness, that TCNets can be applied to the study of Communicating Finite State Machines (using communication through a trace channel), and that they define a hierarchy of partial order semantics for Nets.
Keywords: Mazurkiewicz traces, Fifo Nets, Coloured Nets, concurrency, asynchronous communication, concurrent automata, recognisability.

1 Introduction

Partial orders and "true concurrency" models have been widely generalised in the last two decades for modeling distributed and/or concurrent processes, while their relationships with Petri Nets Theory played a central role ([BC] [BD] [Gi] [Gr] [K1] [NSW] [Pr] [Vo]). Among the many variations on Petri Nets, the ones concerning the structure of arcs labels, tokens and markings led to important developments [Je]. In the present paper, we define a new class of nets, Trace Channel Nets, TCNets in short, where the arcs labels and the markings of the places are traces of a concurrent alphabet [Ma]. They can be viewed as bridging the gap between Simple Coloured Nets, where markings are unordered multisets of colours, and Fifo Nets [FM], where markings are totally ordered multisets.

While Fifo Nets and Communicating Finite State Machines have been used to model processes with asynchronous communication and with total orders on messages, Trace Channel Nets (and Partial Order Channel Nets, see [FB] for the original report) propose in particular a good model for Partial Order Connections and Partial Order Protocols, developed for multimedia applications [ACC][CDL]: the trace or partial order structure of a channel's marking models any precedence or causality relation between alphabetic elementary components, like the partial order on the messages of a POC. Furthermore it acts as a concurrent behaviour "specification" for the output transitions.

Since the late 80's, different formal language theory approaches and results have been successfully extended or adapted to trace languages, [Oc][GPZ], and

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 304–323, 1999.
© Springer-Verlag Berlin Heidelberg 1999

more recently, to pomsets languages [Dr][LW]. Trace Channel Nets can be comprehensively seen as a 'natural' generalisation of existing classes of Nets, corresponding to the extension from words to traces, and benefit from these advances. In Fifo Nets the firing of a transition truncates a prefix of the words marking its input places (the label of the corresponding input arc), and appends suffixes (the words labelling the output arcs), to the markings of its output places. Trace Channel Nets generalise this paradigm: arcs are labelled by traces on a concurrent alphabet, and the notions of trace prefix, residue and concatenation are used to define the TCNet's firing rules (instead of their 'word' versions).

A concurrent alphabet is a pair (A, I) where I is a binary irreflexive relation on an alphabet A, called independence relation. A trace is an equivalence class of words of A^* obtained by permutations of independent letters. The quotient set denoted $(A/\approx_I)$ is a cancellative quotient monoid of A^* called the trace monoid of (A, I). The intuition of the semantics of TCNets is that when two transitions t_1 and t_2 input the letters a and b into the same channel, the order of their firings will be relevant for the resulting marking only if the letters are dependent in the alphabet. If t_1 and t_2 output the letters a and b from a channel, the markings of the channel induce an order on the firings of t_1 and t_2 only if the two letters are dependent.

A first application of TCNets concerns Communicating Finite State Machines. CFSMs use fifo channels to exchange messages, and are a typical case of Fifo Nets. Their extension to trace channels is natural: two safe nets share a trace channel, the first one may input into the channel, the second one may output from it. While the loss of messages has been introduced in CFSM by [CFP], inducing easier verification, Trace Channels relax the total order condition. We give a sufficient condition preserving the rationality of the languages of the two communicating nets, when the channel is taken into account. This condition depends on the rational structure of these languages, and on the labels of the arcs adjacent to the channel, taking into account the dependence relation of the alphabet. It enlights the existence of minimal dependences satisfying the condition, i.e. preserving the rationality of the languages; we think that such a property is typical of the interest of TCNets.

We define the concurrent semantics of TCNets by means of deterministic stably concurrent automata [DK]. They are deterministic transition systems having "local" concurrency relations, defined for each marking on the set of transitions firable from it, and satisfying "good" properties (the diamond, the cube and the inverse cube axioms). The advantages of this definition is that we gain for TCNets the properties worked out for this type of automata, in particular the ones relating recognisable and co-rational sub-languages when the automaton is finite [Dr], i.e. when the net is bounded. Besides the concurrent firings of transitions with no common adjacent places, the trace structure of markings gives the possibility to fire concurrently two transitions that are output of the same place, provided that they consume two independent prefixes of the place marking. The concurrency relations on transitions depend on the independence between the letters of the trace alphabet. If we relax dependence on letters, we may add new

behaviours to the net, but also increase the degree of concurrency in the existing ones.

Independence relations on an alphabet are ordered by inclusion and have a maximal and a minimal element. Simple Coloured Nets and Fifo Nets can be considered as two extreme cases of TCNets, corresponding to these particular relations. Starting from a Fifo Net and considering the different independences on the token alphabet, we define a full partial order of semantics materialised by automata morphisms from the ones with lower independences towards the more concurrent ones. Furthermore, considering the independences from the point of view of the induced behaviours, some of them appear as maximal for the boundedness of the net, as we mentioned above for CFSMs.

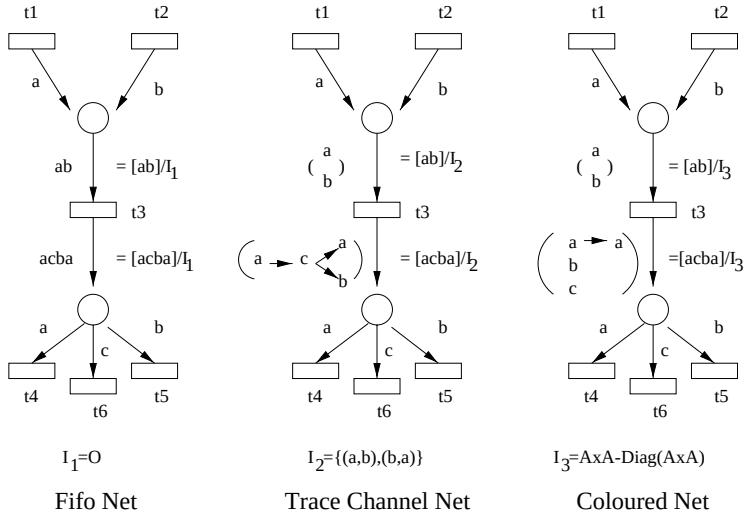


Figure 1.

The three TCNets of the figure 1 share the same structure and the same alphabet with different independence relations. The transition t_3 may occur in the Fifo Net net only after the firing of the sequence t_1t_2, while it may occur in the other ones after any firings of t_1 and t_2. After the firing of t_3, the Fifo Net may only fire the sequence $t_4t_6t_5t_4$. The increasing independence relations from left to right induce new firing sequences for the transitions t_4, t_5 and t_6, say $t_4t_6t_4t_5$ for the middle net,and $t_4t_4t_5t_6$ for the net on the right . The concurrent semantics of section 6 is based on equivalence classes of firing sequences. The two sequences $t_4t_6t_4t_5$ and $t_4t_6t_5t_4$, both enabled in the middle net after the firing of t_3, are equivalent, and compose a partial word on t_4, t_5 and t_6, in which the firings of t_4 and t_5 after the one of t_6 are concurrent.

Related works. If we restrict to Net Theory, besides the above mentioned papers, we can point out Coloured Fifo Nets as defined in [BG] where two types of places are used in Nets, i.e. Fifo places holding words, and bag places holding

multisets of colours. In [F1], the two types of places were defined, but using monochrome tokens for the bags places.

Due to the size specifications, all proofs are omitted and can be found in [F2].

2 Traces

A **trace (or concurrent) alphabet** is a pair (A, I) where A is a set, and I a binary symmetric irreflexive relation on A called the **independence** relation. Its complement $D = A \times A - I$, which is symmetric and reflexive, is called the **dependence** relation.

The **Trace monoid** $(A/\approx_I)$(denoted usually $M(A, I)$, but the letter M is already used for markings) is the quotient of the free monoid A^* by the congruence $\approx_I$, the congruence w.r.t. concatenation generated by the set of pairs $\{(ab, ba) \in A^* \times A^* | (a, b) \in I\}$. Let u be a word in A^*, we denote by $[u]_I$ the class of u in $(A/\approx_I)$. Elements of $(A/\approx_I)$ are called traces. The **monoidal operation** (concatenation) $\circ$ on traces $[u]_I \circ [v]_I$ is defined by $[u]_I \circ [v]_I = [uv]_I$. The monoid $(A/\approx_I)$ is **cancellative,** i.e. $[u]_I \circ [v]_I \circ [u']_I = [u]_I \circ [w]_I \circ [u']_I \Longrightarrow [v]_I = [w]_I$ for any $u, u', v, w \in A^*$, or equivalently $u_1 \approx_I u_1' \wedge u_2 \approx_I u_2' \wedge u_1 v u_2 \approx_I u_1' w u_2' \Longrightarrow v \approx_I w$ for any $u_1, u_1', u_2, u_2', v, w \in A^*$.

The empty trace $[\varepsilon]_I$ is also denoted ε. The number of occurrences of a letter $a \in A$ in a trace $s = [u]_I$ is $|s|_a = |u|_a$, the alphabet of a trace s is $\alpha(s) = \{a \in A, |s|_a \neq 0\}$, and the size of a trace s is $|s| = \sum_{a \in A} |s|_a$, and $|\varepsilon| = 0$. The multiset $[s]$ of a trace s is $[s] \in \aleph^A$such that $\forall a \in A$, $[s](a) = |s|_a$ ($\aleph$denotes the set of natural numbers).

A trace $[u]_I$ is a **prefix** of a trace $[v]_I$, denoted $[u]_I \preceq [v]_I$, by the definition $[u]_I \preceq [v]_I \Longleftrightarrow \exists w \in A^*, [v]_I = [uw]_I$. The **set of prefixes** of a (trace) language L is denoted $Pref(L)$. If $[u]_I \preceq [v]_I$, we define the **residue** $[v]_I - [u]_I$ as follows: $[v]_I - [u]_I = [w]_I \Longleftrightarrow [v]_I = [uw]_I$. The residue is uniquely defined due to the cancellativity of $\circ$. If $\{s_i\}_{i \in J}$ is a finite indexed set of traces such that $i \neq j \Longrightarrow \alpha(s_i) I \alpha(s_j)$, i.e. any two traces have independent alphabets, then we denote by $\bigcirc_{i \in J} s_i$ the trace concatenation of the elements of $\{s_i\}_{i \in J}$, which is independent of the order of concatenations.

A trace $[u]_I$ is **connected** iff it cannot be decomposed into independent factors, i.e. it cannot be written $[u]_I = [vw]_I$ such that $[v]_I \neq \varepsilon \neq [w]_I$ and $\alpha([v]_I) \times \alpha([w]_I) \cap D = \emptyset$. A connected trace $[v]_I$ is a **connected component** of a trace $[u]_I$ iff $[u]_I = [vw]_I$ for some w such that $\alpha([v]_I) \times \alpha([w]_I) \cap D = \emptyset$. The co-star iteration of a trace language L is defined by $L^{co-*} = Comp(L)^*$where $Comp(L)$ is the set of connected components of elements of L. A subset of a monoid (resp. trace monoid) is **rational (resp. co-rational**) iff it is generated from finite subsets by finite applications of concatenation, union and star (resp. co-star) operations. An **iterative factor** of a language (trace language) L is a word (a trace) t such that $ut^*v \subseteq L$ for some words u, v. A language $L \subseteq (A/\approx_I)$ is **star-connected** iff every iterative factor is connected for the dependence relation D. A subset of a monoid is **recognisable** iff it is saturated by a finite index monoid congruence. Ochmanski's theorem states in particu-

lar that a language $L \subseteq (A/\approx_I)$ is star-connected iff it is recognisable iff it is co-rational [Oc].

3 TCNets Structure and Interleaving Semantics

This section presents the basic definitions of TCNets, i.e elementary firing rules and related objects. The definitions are very similar to those of Fifo Nets, except the change from the free monoid A^* to the trace one $(A/\approx_I)$.

3.1 TCNets Structure

A **TCNet** is a tuple $N = (A, I, C, T, W)$ where :

1. (A, I) is a trace alphabet,
2. C, T, are disjoints finite sets. C is the set of channels or places, T the set of transitions,
3. W is the flow relation: $W : C \times T \cup T \times C \longrightarrow (A/\approx_I)$.

A **marking** M is a map from channels to traces : $M : C \longrightarrow (A/\approx_I)$.
The **pre-set** (input set) and the **post-set** (output set) of $x \in C \cup T$ are ${}^\bullet x = \{y \in C \cup T, W(y,x) \neq \varepsilon\}$ and $x^\bullet = \{y \in C \cup T, W(x,y) \neq \varepsilon\}$.

Channel alphabets. The input (resp.output) alphabet of a channel c denoted $\alpha_i(c)$ (resp.$\alpha_o(c)$), is the set of letters in the labels of its input (resp.output) arcs: $\alpha_i(c) = \bigcup\{\alpha(W(t,c)), t \in T\}$ (resp. $\alpha_o(c) = \bigcup\{\alpha(W(c,t)), t \in T\}$). The alphabet of c is $\alpha(c) = \alpha_i(c) \cup \alpha_o(c)$.

Normalised Nets. If the input and output alphabets of c coincide, i.e. $\forall c \in C,\ \alpha_i(c) = \alpha_o(c) = \alpha(c)$ the net is said to be normalised .

Local alphabets. A particular form of TCNets arises where the alphabets of channels are disjoint, i.e. $c \neq c' \Longrightarrow \alpha(c) \cap \alpha(c') = \oslash$, with "local" independences. Each channel has its own concurrent alphabet $(\alpha(c), I(c))$, and this case is easy to achieve by mappings $\alpha(c) \longrightarrow \alpha(c) \times \{c\}$. Except in the section 7, where we consider explicitly local alphabets, the rest of the paper does not depend on this "localisation" property.

We also distinguish semi-alphabetic and alphabetic channels: a channel c is **in-alphabetic**, (resp **out-alphabetic**) iff its input arcs (resp. its output arcs) are labelled by at most single letters, i.e $\forall t \in T, |W(t,c)| \leq 1$ (resp. $\forall t \in T, |W(c,t)| \leq 1$). It is alphabetic iff it has both properties.

Two particular cases correspond to the minimal (empty) and maximal independence relations:

Fifo Nets coincide with the TCNets where the independence relation is empty. If $I = \oslash$, the flow relation is then valued on words : $W : C \times T \cup T \times C \longrightarrow A^*$. All the usual definitions and properties of Fifo Nets are those of TCNets with empty independence.

Simple Coloured Nets are tuples $N = (A, C, T, W, l)$, where the flow function maps arcs on multisets of colours, i.e. $W : C \times T \cup T \times C \longrightarrow \aleph^A$. The **maximal independence** relation for a trace alphabet is $I_m = A \times A - \triangle(A)$, due to its irreflexivity ($\triangle(A)$ is the diagonal of the product). A TCNet with the

maximal independence I_m behaves like a Simple Coloured Net for the interleaving semantics: it is equivalent in a marking to have a bag containing 2 a's or a trace with a connected component $a.a$. For the concurrent semantics this is not the case because in the first case (2 a's) a transition taking one a may fire twice concurrently, which cannot occur in the second. The section 7.4 aggregates Simple Coloured Nets to TCNets w.r.t. concurrency.

3.2 Interleaving Semantics

The interleaving semantics is based on individual transition firing rules, using prefix, residue and concatenation on the trace monoid $(A/\approx_I)$. The left and right cancellativity of $(A/\approx_I)$ are used in the section below.

Let $N = (A, I, C, T, W, l)$ be a TCNet. We say that a transition t **is enabled** at a marking M, denoted $M[t>$, iff it satisfies:

$$M[t> \iff \forall c \in C, W(c,t) \preceq M(c)$$

If t is enabled at M, the firing of t **leads to a new marking M'**, denoted $M[t > M'$, defined as follows:

$$M[t > M' \iff M[t > \bigwedge \forall c, M'(c) = (M(c) - W(c,t)) \circ W(t,c)$$

The definition of the new marking can be expressed in the following way, due to properties of traces (see section 2):

$$M[t > M' \iff M[t > \bigwedge \forall c, W(c,t) \circ M'(c) = M(c) \circ W(t,c)$$

A **sequence of transitions** $\theta = t_1 t_2 ... t_n$ is **enabled** at marking M, and leads to marking M', denoted $M[\theta > M'$, if and only if there is a sequence of markings $M = M_0, M_1, M_2 ... M_n = M'$ such that $M_{i-1}[t_i > M_i \; for \; i = 1, .., n$.

The set of **firing sequences** of a marked net (N, M) where M is a marking of N, is $FS(N, M) = \{\theta \in T^*, M[\theta >\}$

The set of **reachable markings** is $R(N, M_0) = \{M : C \to (A/\approx_I) | \exists \theta \in FS(N, M_0), M_0[\theta > M\}$

A channel $c \in C$ is **bounded** in (N, M_0) iff $\exists n \in \aleph, \forall M \in R(N, M_0)\,, |M(c)|) \leq n$. The marked net (N, M_0) is bounded iff all its channels are bounded.

4 Channel Languages

In this section, we focus on a single channel in the net. The labels of its input and output arcs induce specific structure on its markings, and relations between the sequences of input and output transitions. In particular the balanced language of a net w.r.t. a channel models exactly the constraints induced by the channel on the behaviour of the net.

4.1 Input and Output Traces of Channels

We extend the flow function W to sequences of transitions. For a channel c, W yields two monoid morphisms $W(_, c) : T^* \longrightarrow (A/\approx_I)$ and $W(c,_) : T^* \longrightarrow (A/\approx_I)$, defining the input and output traces of the channel c for a given word $\theta = t_1 t_2 ... t_n \in T^*$:

$$W(\theta, c) = W(t_1, c) \circ W(t_2, c) \circ .... \circ W(t_n, c)$$

$$W(c, \theta) = W(c, t_1) \circ W(c, t_2) \circ ..... \circ W(c, t_n)$$

We use the notation $W(_, c)(L) = \{W(\theta, c), \theta \in L\}$.

In the examples of the figure 1, if we let $\{c_1\} = t_1^\bullet$, we have $W(t_1 t_2 t_1, c_1) = [aba]_I$, with $I = I_1, I_2$ or I_3 depending on the chosen net.

By induction on the length of a firing sequence, we prove the following properties.

Lemma 1. *Let be (N, M) is a marked TCNet,then*

1. $\forall \theta \in FS(N, M), \forall c \in C,\ W(c, \theta) \preceq M(c) \circ W(\theta, c)$.
2. $M[\theta > M' \Longrightarrow \forall c \in C,\ M'(c) = (M(c) \circ W(\theta, c)) - W(c, \theta)$.

4.2 Balanced Languages w.r.t. Channels

We characterise the constraints induced by a single channel on the net behaviours. For that purpose, we consider a net N, a channel c, and we define the hiding of c in N, $N \setminus c$, as the net where the channel c and its adjacent arcs have been suppressed. We show that the firing sequences of N are the ones of $N \setminus c$ which we call balanced w.r.t. c, i.e. such that for any of its prefixes, the corresponding output from c is a trace prefix of its inputs in c.

Let c be a channel, M a marking, the **balanced language of N w.r.t. c and M** is

$$BL(c, M) = \{\theta \in T^* | \forall \nu \in T^*,\ \nu \preceq \theta \Longrightarrow W(c, \nu) \preceq M(c) \circ W(\nu, c)\}$$

We then define the "hiding" of a channel c in a net N, denoted $N \setminus c$ and then compare its firing sequences with the ones of the complete net.

Let be a net $N = (A, I, C, T, W, l)$, the **hiding of c in N** is the net

$N \setminus c = (A, I, C - \{c\}, T, W \setminus c, l)$

where $W \setminus c$ is the restriction of the flow mapping W to the set $((C - \{c\}) \times T) \cup (T \times (C - \{c\}))$.

We denote by $M \setminus c$ the restriction of a marking M to $C - \{c\}$. The firing language of N is the intersection of the firing language of $N \setminus c$ and the balanced language of N w.r.t. c, with the hypothesis that ${}^\bullet c \cap c^\bullet = \oslash$, i.e. no transition is both an input and an output for c.

Proposition 1. *(Balanced Language).*

If c is a channel such that ${}^\bullet c \cap c^\bullet = \oslash$, then:

$$FS(N, M) = FS(N \setminus c, M \setminus c) \cap BL(c, M)$$

4.3 Relations between Inputs and Outputs in Balanced Languages

We find interesting for understanding TCNets to characterise algebraically the relationships between the projections of a balanced language on the sets ${}^\bullet c$ and $c^\bullet$ of input and output transitions, i.e. $BL(c,M)/{}^\bullet c$ and $BL(c,M)/c^\bullet$. The following lemma may seem verbose, but facilitates the next sections.

Lemma 2. *If* ${}^\bullet c \cap c^\bullet = \oslash$, *then*

1. *If* $\theta \in ({}^\bullet c)^*$ *and* $\theta' \in (c^\bullet)^*$ *then* $\theta\theta' \in BL(c,M) \Longleftrightarrow W(c,\theta') \preceq M(c) \circ W(\theta,c)$
2. *If* $\theta \in ({}^\bullet c)^*$ *then* $\{\theta' \in (c^\bullet)^*, \theta\theta' \in BL(c,M)\} = W(c,_)^{-1}(Pref(M(c) \circ W(\theta,c)))$
3. *If* $\theta \in (c^\bullet)^*$ *then* $\{\theta' \in ({}^\bullet c)^*, \theta'\theta \in BL(c,M)\} = W(_,c)^{-1}((W(c,\theta) \circ (A/_{\approx_I})) - M(c))$
4. $\theta \in BL(c,M) \Longrightarrow (\theta/{}^\bullet c)(\theta/c^\bullet) \in BL(c,M)$

We associate to any sequence of input transitions θ, the set $En(c,\theta,M)$ of sequences of output transitions θ' such that $\theta\theta' \in BL(c,M)$, and which we call **enabled** after θ:

$En(c,_,M) : ({}^\bullet c)^* \longrightarrow \wp((c^\bullet)^*)$ with $En(c,\theta,M) = \{\theta' \in (c^\bullet)^*, \theta\theta' \in BL(c,M)\}$

In the same way we can define a **requirement** mapping :

$Rq(c,_,M) : (c^\bullet)^* \longrightarrow \wp(({}^\bullet c)^*)$ with $Rq(c,\theta,M) = \{\theta' \in ({}^\bullet c)^*, \theta'\theta \in BL(c,M)\}$.

A natural question that arises is: if $L \subseteq ({}^\bullet c)^*$ is a recognisable language, at which conditions is $En(c,L,M) \subseteq (c^\bullet)^*$ a recognisable language ? The proposition below gives a sufficient condition, based on the image by $W(_,c)$ of the iterative factors of L. The proposition relies on Ochmanski's theorem and the following fact:

If L is a rational language, $L \subseteq T^*$, and $\varphi : T^* \longrightarrow (A/\approx_I)$ a monoid morphism, then $\varphi(L)$ is co-rational if and only if the image of any iterative factor of L by φ is connected in $(A/\approx_I)$.

Proposition 2. *(Rationality of* $En(c,L,M)$*).*
If $L \subseteq ({}^\bullet c)^*$ *is a rational language such that for each iterative factor* F *of* L, $W(_,c)(F) \subseteq (A/\approx_I)$ *is connected, then* $En(c,L,M) \subseteq (c^\bullet)^*$ *is rational.*

Note that this condition is not necessary as shown in the example in figure 2.d of section 5.

5 Bounded Petri Nets Communicating through Trace Channels

Communicating Finite State Machines use Fifo channels to exchange messages, and are a typical case study for Fifo nets. Their extension to trace channels is natural. We only study communication through a single channel, and throughout this section we consider a TCNet $N = (A, I, C, T, W, l)$ and a channel $c \in C$ such that $N \setminus c$ has two disjoint connected components $N_i = (A, I, C_i, T_i, W_i, l_i), i = 1, 2$, such that ${}^\bullet c \subseteq T_1$ and $c^\bullet \subseteq T_2$. If M is a marking of N, we denote by M_1 and M_2 its restrictions to C_1 and C_2.

5.1 Components Languages

We suppose that the nets (N_i, M_i) are bounded, and so, $FS(N_i, M_i)$,$i = 1, 2$, are rational languages in T_i^*. The problem we address first is to find sufficient conditions on the net such as to ensure the rationality of the projections $FS(N, M)/T_i$, $i = 1, 2$. As far as we have $FS(N, M)/T_1 = FS(N_1, M_1)$, the question is : when is $FS(N, M)/T_2$ rational? The proposition below relies on proposition 2, and considers the image by the input flow function $W(_, c)$ of the iterative factors of $FS(N_1, M_1)$.

Proposition 3. *(Rationality of Projected Languages).*
If the sets $FS(N_i, M_i), i = 1, 2$, are rational languages, and if for every iterative factor F of $FS(N_1, M_1)$, $W(_, c)(F)$ is connected, then $FS(N, M)/T_i, i = 1, 2$. are rational languages.

Note that the condition of the proposition is valid for some initial marking of c iff it is valid for any initial marking of c.

5.2 Empty Channel Languages

Emptiness of the communication channel is an important property, used in particular in the definition of half-duplex systems. We focus now on the firing sequences which preserve the emptiness. For this purpose, we define the **empty channel language** of (N, M) as $L_{c,\varepsilon} = \{\theta \in FS(N, M), W(\theta, c) = W(c, \theta)\}$. If we suppose that $M(c) = \varepsilon$, we have
$L_{c,\varepsilon} = \{\theta \in FS(N, M), M[\theta > M' \Longrightarrow M'(c) = \varepsilon\}$.
With the hypothesis that $FS(N, M)/T_i, i = 1, 2$ are rational languages, i.e. a property ensured by the previous proposition, and using the same type of arguments, we give a sufficient condition to ensure the rationality of the projections $L_{c,\varepsilon}/T_1$ and $L_{c,\varepsilon}/T_2$. The same type of argument as in the previous proposition are used, but in this case we consider also the iterative factors of the receiving net language.

Proposition 4. *Let $L_i = FS(N, M)/T_i, i = 1, 2$ be rational, then if for every iterative factor F of L_1, $W(_, c)(F)$ is connected, and if for every iterative factor F of L_2, $W(c, _)(F)$ is connected, then $L_{c,\varepsilon}/T_i, i = 1, 2$, are rational languages.*

The connectivity of $W(_, c)(F)$ and $W(c, _)(F)$ used in the two propositions above depends on the relation I. If I is too large, the conditions for the rationality of the languages of the component nets are not verified. More generally in section 6, we will show the existence of maximal independences on channels alphabets w.r.t. the rationality of the firing language of a net.

5.3 Examples

The figure 2 shows four TCNets, each one composed of a pair of disjoint component nets, denoted N_1 and N_2, which communicate through a trace channel. The nets in 2.a and 2.b have bounded components, and the proposition 3. applies. We can ensure the rationality of the projected behaviours.

In the net 2.a, we have $FS(N_1) = Pref((t_1t_2)^*)$, then t_1t_2 is the only iterative factor of the input language, and we need to ensure that $W(_, c)(t_1t_2) = [abcd]_I$ is connected while $W(t_i, c), i = 1, 2$ do not need to be connected. This is the case for $I = \{(a, b), (b, a), (a, c), (c, a), (b, c), (c, b)\}$, and the projected behaviours on T_2 are $FS(N)/T_2 = Pref((Perm(t'_1.t'_2.t'_3).t'_4)^*) \cap FS(N_2)$ (the iterative factor is any permutation of the sequence $t'_1.t'_2.t'_3$ followed by t'_4).

In the net 2.b, $FS(N_1) = Pref(t_1^*t_2^*)$ has two iterative factors, t_1 and t_2, and the proposition applies only if $W(t_i, c), i = 1, 2$ are connected, i.e $\{(a, b), (c, d)\} \subset D$. If $I = (\{a, b\} \times \{c, d\}) \cup (\{c, d\} \times \{a, b\})$, then $FS(N)/T_2 = Pref((t'_1.t'_2.)^* ||| (t'_3.t'_4)^*) \cap FS(N_2)$ which can be expressed as a rational expression (the shuffle of rational languages is rational).

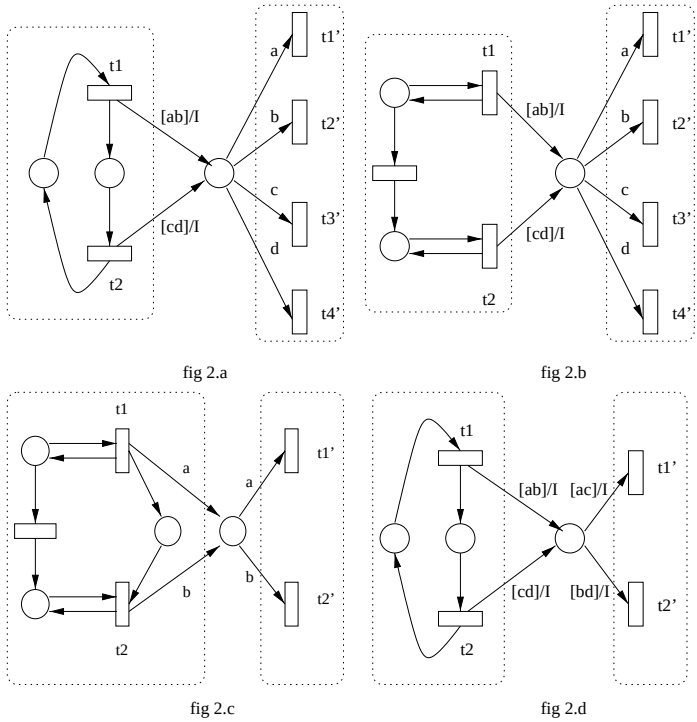


Figure 2.

The net 2.c is a counter-example to the rationality of $FS(N_1)/^\bullet c$, $FS(N_1)/^\bullet c = \{t_1^n t_2^n, n \in \aleph\}$, and $\{W(\theta, c) | \theta \in FS(N)\} = \{[a^n b^n]_I, n \in \aleph\}$, which cannot be rational, independently from I.If aIb, $FS(N)/T_2 = \{\nu \in T_2^*, |\nu|_{t'_1} = |\nu|_{t'_2}\} \cap FS(N_2)$.

The net 2.d. shows that the condition of the proposition 3 is not necessary: If a and b are independent from c and d, with aDb and cDd, then $W(_, c)(t_1t_2) = [abcd]_I$ is not connected but the induced behaviour on T_2 is rational : $FS(N)/T_2 = Pref((t'_1t'_2)^*) \cap FS(N_2)$.

6 Concurrent Semantics

6.1 Modeling by Automata with Concurrency Relations

We define the concurrent semantics for TCNets, by **deterministic automata with concurrency relations** [DK]. For each TCNet N, we define an automaton $Aut(N)$, and we show that $Aut(N)$ is **stably concurrent**. This type of automata has many advantages:

- Equivalence classes of sequential behaviours correspond to maximally concurrent firable partial orders of transitions [BDK][Vo]. We do not need to define explicitly firing rules for steps of transitions or partial words of transitions.
- If the net is bounded, the automaton is finite, and co-rationality and recognisability are related for the sub-languages of the automaton [Dr].
- A simulation relation between TCNets can be defined by means of automata morphisms, which take concurrency of behaviours into account. These morphisms relate nets with the same structure but different dependence relations on the tokens alphabet, and particular dependences can be underscored in this framework.

In this section we need to distinguish the transitions of the net which coincides with the *events* of the automaton, from the *transitions* of the automaton, i.e. triples composed by a source marking, an event and a target marking. The **concurrency relations on the events** of the automaton, denoted $\|_M$, depend on the marking M where the events are enabled and on the independence relation I. More precisely two events are concurrent at a marking M iff they are both enabled at M, and they neither both output dependent letters from the same channel, nor both input dependent letters in the same channel. This is formalised in the item 4 of the definition below.

The automaton of a TCNet $N = (A, I, C, T, W, l)$, denoted $Aut(N)$, is the tuple

$Aut(N) = (Q, E, \Theta, \{\|_M\}_{M \in Q})$ where

1. $Q = \{M : C \longrightarrow (A/\approx I)\}$ is the **set of states**, coinciding with the markings of N.
2. $E = T$, the **set of events** coincides with the set of transitions of N.
3. $\Theta \subseteq Q \times E \times Q$ is the **transition set** , defined by $(M, t, M') \in \Theta \Longleftrightarrow M[t > M'$.
4. $\|_M \subseteq E \times E$, defined by $t \parallel_M t' \Longleftrightarrow M[t > \bigwedge M[t' > \bigwedge tI_E t'$, is the **concurrency relation**. The auxiliary relation on events I_E does not rely on M :

$tI_E t' \Longleftrightarrow$

$$\forall c \in C,\ \alpha(W(c,t)) \times \alpha(W(c,t')) \cap D = \oslash \bigwedge \alpha(W(t,c)) \times \alpha(W(t',c) \cap D = \oslash$$

We recall that D is the complement of I in $A \times A$.

Note that if the pre and post sets of two transitions are disjoint, they are related by I_E: $t^\bullet \cap t'^\bullet = \oslash \bigwedge {}^\bullet t \cap {}^\bullet t' = \oslash \Longrightarrow tI_E t'$.

Note also that for Fifo Nets, the two properties are equivalent, and in that case two transitions can be concurrent only if they do not share some input or output channel: $I = \oslash \Longrightarrow (t^\bullet \cap t'^\bullet = \oslash \bigwedge^\bullet t \cap^\bullet t' = \oslash \Longleftrightarrow tI_E t')$

To satisfy the definition of automata with concurrency relations, we have to verify the so-called (forward) **diamond** property, which ensures that the order in which we fire concurrent events is irrelevant w.r.t. the obtained marking.

Note that the proofs of the three lemmas below rely on the fact that the dependence relation D is reflexive, and for that reason this construction cannot be used to give a concurrent semantics to Coloured Nets. We shall overcome this restriction in the section 7.4.

Lemma 3. *(Diamond Property).*

$$t \parallel_M t' \Longrightarrow \exists M',\ M[t.t' > M' \bigwedge M[t'.t > M'$$

Note that the reverse implication $(\exists M',\ M[t.t' > M' \bigwedge M[t'.t > M') \Longrightarrow t \parallel_M t'$ does not hold; the left hand side may be due to $W(c,t) = W(c,t')$ or $W(t,c) = W(t',c)$ for some $c \in C$, and we do not consider in that case the transitions to be concurrent.

6.2 Stability Property

The automaton $Aut(N)$ is in fact a deterministic **stably concurrent automaton** because it is deterministic (obvious from definitions) and it satisfies the following cube and inverse cube properties [DK][Dr].

Lemma 4. *(Cube Property).*
Let be $M \in Q$, and three events t_1, t_2, t_3 such that $t_1 \parallel_M t_2$, $t_2 \parallel_M t_3$, and $t_1 \parallel_{M_2} t_3$, then $t_1 \parallel_M t_3$, $t_2 \parallel_{M_1} t_3$, and $t_1 \parallel_{M_3} t_2$, where $M[t_i > M_i, i = 1,2,3$.

Lemma 5. *(Inverse Cube Property).*
Let be $M \in Q$, and three events t_1, t_2, t_3 such that $t_1 \parallel_M t_3$, $t_2 \parallel_{M_1} t_3$, and $t_1 \parallel_{M_3} t_2$, then $t_1 \parallel_M t_2$, $t_2 \parallel_M t_3$, and $t_1 \parallel_{M_2} t_3$ where $(M[t_i > M_i, i = 1,2,3)$.

A stably concurrent automaton being characterised by the cube and inverse cube properties, the following proposition resumes the three lemmas above:

Proposition 5. *The automaton $Aut(N)$ of a TCNet N is a **deterministic stably concurrent automaton**.*

6.3 Computations Sequences and Concurrent Behaviours

The difference between firing sequences and computation sequences is that the latter record the intermediate markings. A **computation sequence** is either the empty one (noted ε) or a finite sequence $u = u_1, u_2, ..u_n$, where $u_i = (M_{i-1}, t_i, M_i), i = 1, n$. M_0 and M_n are the domain $dom(u)$ and codomain $cod(u)$ of u. The sequence $evseq(u) = t_1 t_2 .. t_n$ is a firing sequence of $FS(N, M_0)$. The

set of computation sequences of $Aut(N)$ is denoted $CS(N)$, and is a partial monoid, where the concatenation of u and v is defined iff $dom(v) = cod(u)$. We define on the monoid of computation sequences a congruence denoted $\sim$, generated by the **permutation of concurrent transitions**:

$$(M, t_1, M_1)(M_1, t_2, M') \sim (M, t_2, M_2)(M_2, t_1, M') \Longleftrightarrow t_1 \parallel_M t_2$$

extended to a congruence w.r.t. the partial concatenation. Note that two equivalent sequences have the same domain, codomain and event sets.

The set of **concurrent behaviours** is the quotient $Beh(N) = CS(N)/\sim \cup\{0\}$, where 0 stands for the undefined concatenation and is absorbing. As traces in standard trace monoids, the concurrent behaviours in $Beh(N)$ can be represented by partial words of transitions, using the same construction. Note that we do not consider infinite sequences or computations in the present work.

In the original work [FB], we defined a concurrent semantics of TCNets based on steps and partial words firing rules, following [Vo]. The one presented here yields maximally concurrent behaviours, while the previous one was a "weak" semantics, where any increment of the partial order between events in a behaviour yields another admissible behaviour, allowing the firing of the partial word $t \to t'$ at a marking M where the concurrent step $t \, co \, t'$ is enabled. The two semantics coincide on maximal concurrency.

Marked nets and automata with initial state.

If we consider a marked TCNet (N, M_0), then the **automaton with initial state** $Aut(N, M_0)$is the tuple

$$Aut(N, M_0) = (Q, E, \Theta, \{\parallel_M\}_{M \in Q}, M_0)$$

where the set of states is $Q = R(N, M_0)$, and the other components are the same as in $Aut(N)$.

Note that this automaton has the same cube and inverse cube properties as $Aut(N)$. We denote by $CS(N, M_0)$ and $Beh(N, M_0)$ the sets of computation sequences and concurrent behaviours of $Aut(N, M_0)$, i.e. the computation sequences (and concurrent behaviours) of $Aut(N)$ having as domain the initial state M_0.

7 From Fifo Nets to Simple Coloured Nets

We explicitate in this section the **partial order of semantics** induced by the different independence relations on a single net structure. We relate in this section TCNets with the same alphabets, sets of channels and transitions that differ only by the independence relations and flow functions. For simplicity, we **do not work up to nets isomorphisms**, in particular transitions or channels renamings (they do not constitute a drawback here, but should be introduced to define operators and observations on TCNets).

The figure 3 below illustrates the main aspects of the section. It represents a trace net N where the independence relation is left undefined, and the spectrum of semantics on the net structure of N, induced by the inclusion order on independence relations.

For the relation $I = \{(a,c),(c,a),(b,c),(c,b),(b,d),(d,b)\}$, the net is bounded, while for $I' = \{(a,c),(c,a),(a,d),(d,a),(b,c),(c,b),(b,d),(d,b)\}$ it is unbounded. In this case for all independences I for which the trace $W(t_1.t_2,c) = [abcd]_I$ is connected, the net is bounded, while the boundedness is lost when $W(t_1.t_2,c)$ is disconnected: in the former case, all transitions t'_1, t'_2, and t'_3 have to fire once between two occurrences of t'_4, inducing a bound on the markings of the channel output of t_1 and t_2. Note the similarity between this particular situation and the properties worked out in the proposition 2 and section 5 about bounded nets communicating through trace channels.

In the following, we first define the quotient net N/I' obtained from the net N by replacing its independence relation I by a relation I' such that $I \subseteq I'$. We show that there is a concurrent automata morphism between $Aut(N)$ and $Aut(N/I')$. The relation between N and $N' = N/I'$ is a partial order denoted $N \sqsubseteq N'$, defined on sets of nets with the same underlying structure, having a set of Fifo Nets as minimal elements, and with a unique maximal element, a quotient N/I_m, where I_m is the maximal independence relation.

In this section, we consider TCNets with disjoint channels alphabets. We could define formally a set of concurrent alphabets $\{(A_c, I_c), c \in C\}$ with $A_c = \alpha(c)$, yielding monoids $(A_c/\approx_{I_c})$. For simplicity and readability, we use (A,I) and $(A/\approx_I)$, and thus we take $I \subseteq I'$ as a shortcut for $I_c \subseteq I'_c$, $\forall c \in C$. Proofs do not suffer from this simplification, all traces being "local" to a channel, and we gain in readability.

7.1 Incrementing Independence

Starting from the definition of a Fifo Net $N = (A, \oslash, C, T, W, l)$, we can define on N a TCNet semantics associated to an independence relation I on A, considering the arcs labels up to the monoid congruence $\approx_I$, and using the firing rules in $(A/\approx_I)$. More generally, let be I and I' two independence relations on A, with $I \subseteq I'$, there is a canonical monoid morphism

$$\varphi_{I,I'} : (A/\approx_I) \longrightarrow (A/\approx_{I'})$$

which maps $(A/\approx_I)$ on $(A/\approx_{I'})$ defined by $\varphi_{I,I'}([u]_I) = [u]_{I'}$. We denote by $\circ_I$ and $-_I$ the trace concatenation and residue in $(A/\approx_I)$. A net $N = (A, I, C, T, W, l)$ can be quotiented by the relation I' as follows:

Let be $N = (A, I, C, T, W, l)$a TCNet, and a independence relation I' on A such that $I \subseteq I'$, the **quotient** N/I' is the trace net $N/I' = (A, I', C, T, W', l)$ where $W' = \varphi_{I,I'} \circ W$.

For convenience we denote by $s_{I'}$ the image $\varphi_{I,I'}(s) \in (A/\approx_{I'})$ of a trace $s \in (A/\approx_I)$.

The main property is that the firing sequences of a net N are preserved in the quotient, N/I' and the images of its reachable markings by $\varphi_{I,I'}$ are reachable markings of N/I'.

Lemma 6. *Let be $N = (A, I, C, T, W, l)$ a TCNet, and $I \subseteq I'$, then*

1. $FS(N, M) \subseteq FS(N/I', M_{I'})$
2. $\varphi_{I,I'}(R(N, M)) \subseteq R(N/I', M_{I'})$.

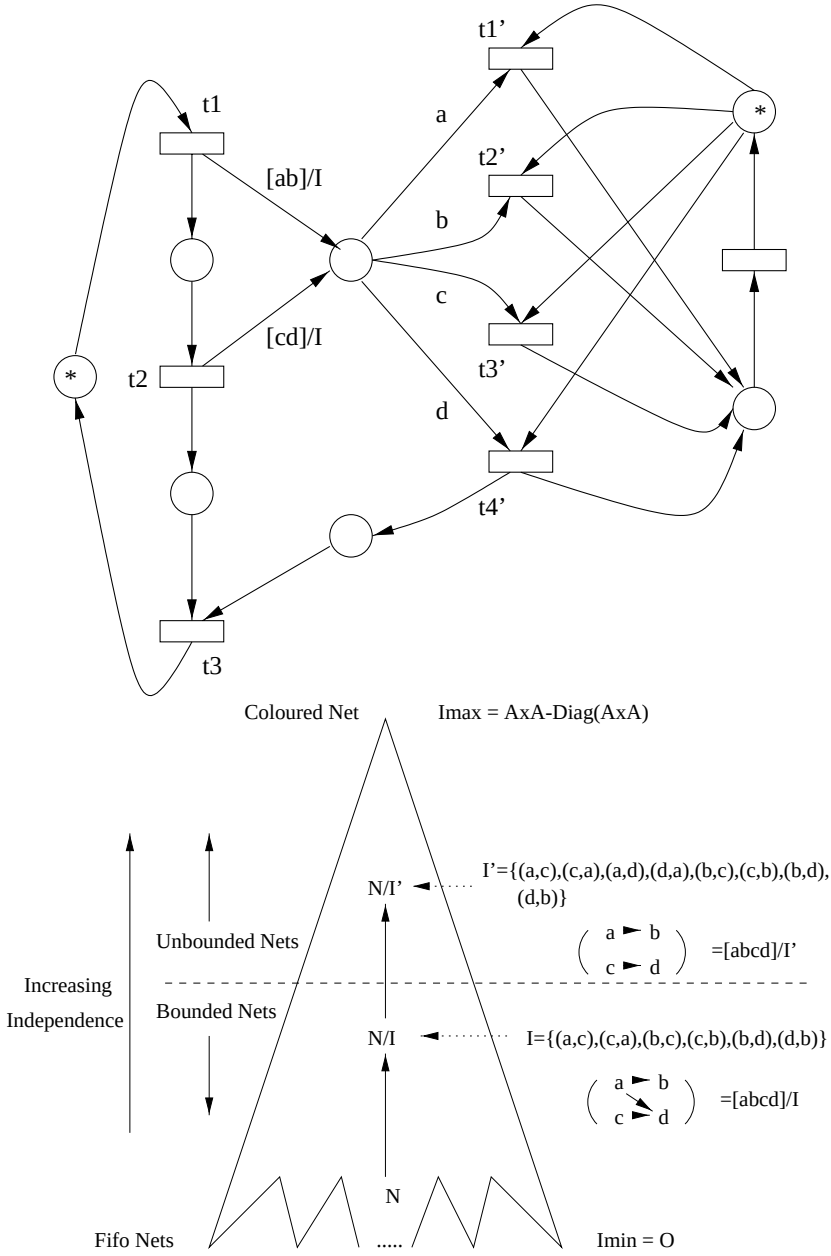

t1'
t1
a
*
t2'
[ab]/I
b
c
[cd]/I
t3'
*
t2
d
t4'
t3
Coloured Net
Imax = AxA-Diag(AxA)
N/I'
I'={(a,c),(c,a),(a,d),(d,a),(b,c),(c,b),(b,d),
(d,b)}
Unbounded Nets
a b
c d
=[abcd]/I'
Increasing
Independence
Bounded Nets
N/I
I={(a,c),(c,a),(b,c),(c,b),(b,d),(d,b)}
a b
c d
=[abcd]/I
N
Fifo Nets
.....
Imin = O

Figure 3.

The relation between $Aut(N)$ and $Aut(N/I')$ is a morphism of concurrent automata, following the definition of [DK]. Such morphisms ensure that the concurrent behaviours of the source net can be executed in the target system in a more (or at least equal) concurrent way.

A **morphism between the automata** $Aut = (Q, E, \Theta, \{\|_M\}_Q, M_0)$ and $Aut' = (Q', E', \Theta', \{\|_M\}_{Q'}, M_0')$ is a pair $(\pi, \eta) : Aut \longrightarrow Aut'$ where

$\pi : Q \longrightarrow Q'$,$\eta : E \longrightarrow E'$, such that:

1. $(M, t, M') \in \Theta \Longrightarrow (\pi(M), \eta(t), \pi(M')) \in \Theta'$
2. $\pi(M_0) = M_0'$
3. $t \parallel_M t' \Longrightarrow \eta(t) \parallel_{\pi(M)} \eta(t')$.

Proposition 6. *Let (N, M_0) be a marked TCNet, with $N = (A, I, C, T, W, l)$, and $I \subseteq I'$, the pair $(\varphi_{I,I'}, id_E)$ is an automata morphism*

$$(\varphi_{I,I'}, id_E) : Aut(N, M_0) \longrightarrow Aut(N/I', M_{0I'})$$

Increasing the independence on letters preserves the firing sequences and may enlarge their equivalence classes w.r.t. the congruence $\sim$, incrementing concurrency. Viewing behaviours as partial words of transitions, we have the relation $Beh(N) \subseteq Aug(Beh(N/I'))$, where Aug operates on its argument by incrementing the order relations of the included partial words.

The morphism may also introduce new transitions, i.e. transitions which are not the image $(M_{I'}, t, M_{I'}^{'}) \in \Theta'$ of a transition $(M, t, M') \in \Theta$. The situation is specified as follows:

$\exists M \in R(N, M_0), t \in T : M_{I'}[t > \bigwedge (\forall M^{'} \in R(N, M_0), M_{I'} = M_{I'}^{'} \Longrightarrow \neg M^{'}[t >)$.

It seems that the ongoing study of the different situations induced by an increment of independence should yield interesting properties and a new insight on net structures.

7.2 The Independence Partial Order

As mentioned previously, the maximal independence for a trace alphabet is $I_m = \{A \times A - \triangle(A)\}$, more precisely $I_m = \cup_{c \in C}(\alpha(c) \times \alpha(c) - \triangle(\alpha(c)))$. We say that two TCNets N and N' are **structure equivalent** , denoted $N \leftrightarrow N'$, iff they have the same quotient by the relation I_m, i.e. $N/I_m = N'/I_m$. The relation $\leftrightarrow$ is obviously transitive and symmetric. We denote by $[[N]]$ the class of TCNets equivalent to N.

The **independence partial order** $([[N]], \sqsubseteq)$ is composed of pairs $N \sqsubseteq N'$ such that N' is a quotient N/I' for some relation I'. It is a transitive relation on TCNets, due to the following property : if $N = (A, I, C, T, W, l)$ and $I \subseteq I' \subseteq I''$, then $N/I'' = (N/I')/I''$. Antisymmetry follows from the one of $\subseteq$. The formal definitions are as follows:

Let be two TCNets $N = (A, I, C, T, W, l)$ and $N' = (A, I', C, T, W', l)$ then

1. $N \sqsubseteq N' \Longleftrightarrow I \subseteq I' \bigwedge N' = N/I'$.
2. $N \leftrightarrow N' \Longleftrightarrow N/I_m = N'/I_m$.
3. $[[N]] = \{N', N \leftrightarrow N'\}$

Lemma 7. *The equivalence $\leftrightarrow$ is the the symmetric transitive closure of $\sqsubseteq$.*

For any Fifo Net N, $[[N]]$ contains all the nets which can deduced from N by permutations on the arcs labels $W(x,y) \in A^*$, and these are the minimal elements of $([[N]], \sqsubseteq)$. The first result of the definition of $([[N]], \sqsubseteq)$ is the existence of particular independences w.r.t. different properties, minimal independences for liveness or reachability properties, maximal ones for boundedness as shown in the next section.

7.3 Critical Independences w.r.t. Boundedness

We consider boundedness property w.r.t. the independence order, and the lemma below states that if N' is bounded and $N \sqsubseteq N'$, then N is bounded too.

Lemma 8. *Let $N = (A, I, C, T, W, l)$be a TCNet, and $I \subseteq I'$, if $(N/I', M_{0I'})$ is bounded, then (N, M_0) is bounded.*

A immediate consequence is that if N is unbounded and $N \sqsubseteq N'$, then N' is unbounded. The lemma above induces the existence of **critical** independences, i.e. maximal w.r.t. the boundedness of a net.

Proposition 7. *(Maximal Independences w.r.t. Boundedness)*
If $N = (A, I, C, T, W, l)$is bounded, and N/I_m is not bounded, then we can find a (non unique) relation I', with $I \subseteq I'$ such that

1. *N/I' is bounded.*
2. *if $I' \subset I''$, then N/I'' is unbounded.*

We can relate this result to the propositions of the section 5, in which the sufficient conditions for the rationality of the components languages yield also maximal independence relations, those ensuring the connectedness of specific traces of $(A/\approx_I)$.

7.4 Simple Coloured Nets

The TCNet N/I_m has a concurrent semantics in terms of stably concurrent automata as defined previously, but this definition forbids concurrent firing of transitions which output a common letter in the same channel. In particular it forbids autoconcurrency of the net transitions. For that reason we have to define specific firing rules and concurrent semantics for a coloured net, and then introduce it in the framework of the present section.

A Simple Coloured Net $N = (A, C, T, W, l)$, has the weight function and the markings valued as multisets on $A : W : C \times T \cup T \times C \longrightarrow \aleph^A$, $M : C \longrightarrow \aleph^A$. The firing rules are defined as follows:

1. $M[t> \iff \forall c \in C, W(c,t)) \leq M(c)$
2. $M[t > M' \iff M[t > \bigwedge \forall c, M'(c) = (M(c) - W(c,t)) + W(t,c)$

where the relation $\leq$ and operators $-$ and $+$ are the standard ones on multisets, i.e. defined componentwise.

The main difference between the automaton with concurrency relations $Aut_{co}(N)$ defined for a Coloured Nets and the ones of TCNets lies in the concurrency relation. Due to this difference, autoconcurrent firings of a transition are now allowed, and the new definition invalidates both the cube and inverse cube properties.

$Aut_{co}(N) = (Q, E, \Theta, \{\|_M\}_{M \in Q})$ where

1. $Q = \{M : C \longrightarrow \aleph^A\}$ is the set of states,
2. $E = T$, the set of events coincides with the set of transitions of N.
3. $\Theta \longrightarrow \subseteq Q \times E \times Q$ is the transition set, defined by $(M, t, M') \in \Theta \Longleftrightarrow M[t > M'$.
4. $\|_M \subseteq E \times E$, defined by $t \|_M t' \Longleftrightarrow \forall c \in C, W(c,t) + W(c,t') \leq M(c)$.

The diamond property of $Aut_{co}(N)$ is obvious.

Let be $u \in (A/\approx_I)$, we denote by $[u] \in \aleph^A$ the multiset such that $\forall a \in A$, $[u](a) = |u|_a$.

To any TCNet, we associate a unique Coloured Net by replacing the flow function by its multisets value.

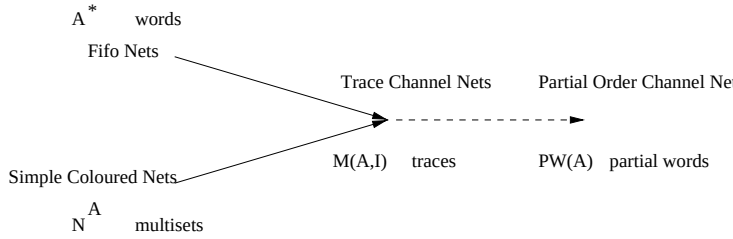
Let be $N = (A, I, C, T, W, l)$ a TCNet, the **coloured net associated** to N, is $N_{co} = (A, C, T, [W], l)$ where $[W](x,y) = [W(x,y)]$ for $(x,y) \in C \times T \cup T \times C$.

Lemma 9. *Let be $N = (A, I, C, T, W, l)$ a TCNet then the pair $([_], id_E)$ is an automata morphism*

$([_], id_E) : Aut(N, M) \longrightarrow Aut_{co}(N_{co}, [M])$

This last section has completed the independence partial order $([[N]], \sqsubseteq)$ by the Simple Coloured Net N_{co}.

8 Conclusion



The diagram above shows the inclusions between four classes of Nets, related to the inclusions of the underlying markings and edge labels sets. In the present paper, we have presented a framework introducing the two arrows on the left.

Although traces can be viewed as particular partial words, an embedding morphism w.r.t. concatenation has not been defined yet, and the arrow on the right is ongoing work.

Trace Channel Nets seem to be a 'sensible' generalisation of two existing and until now unrelated classes of Petri Nets. They induce Net Theory to follow the development of Formal Language Theory from words languages to traces and to partial words languages. Even if they inheritate their Turing power from Fifo Nets, different tractable classes of interesting problems arise from their definition, and hopefully they will result in specification and optimisation techniques for systems of communicating processes. In particular the use of trace channels in CFSMs should result useful in protocols specifications and proofs, to be applied for Partial Order Connections and Protocols. They seem well adapted to represent concurrent systems where data are structured by order or dependence relations, and that is the case when data are themselves behaviours specifications. Notions of observability, equivalence, refinement and implementation, compositionality or modularity should be developed for a practical use of the model, and are the topic of ongoing work.

Acknowledgments

Thanks to Marc Boyer, Michel Diaz and ICATPN referees for clarity improvements. Thanks also to Richard Hopkins for having the idea in 1992 to consider partial orders instead of words in Fifo Nets places of the paper [F1].

References

ACC. Amer.P,Chassot.C,Conolly.T.J,Diaz.M.,Conrad.P. *Partial-order transport service for multimedia and other applications.* IEEE/ACM Transactions on Networking, Vol.2, N5, 440-456, October 1994

BD. Best.E,Devillers.R. *Sequential and concurrent behaviour in Petri Nets theory.* Theoretical Computer Science 55 (1987), 87-136.

BDK. Bracho.P,Droste.M,Kuske.D. *Representation of computations in concurrent automata by dependence orders.* Theoretical Computer Science 174 (1997) 67-96.

BC. Boudol.G,Castellani.I. *On the semantics of concurrency:partial orders and transition systems.* Lecture Notes in Computer Science, Vol. 249; CAAP 1987, 123-137. Springer-Verlag, 1987.

BG. Benalycherif.M,Girault.C. *Behavioural and Structural Composition Rules Preserving Liveness by Synchronisation for Coloured FiFo Nets.* Lecture Notes in Computer Science, Vol 1091; ICATPN 96, 73-92. Springer-Verlag, 1996.

CDL. Chassot.C.DDiaz.M,Lozes.A. *From the partial order concept to partial order multimedia connections.* Journal of High Speed Networks, vol. 5, n. 2, 1996.

CFP. Cece.G,Finkel.A,Purushotaman.S. *Unreliable channels are easier to verify than perfect channels.* Information and computation, vol.124 (1),20-31,1996.

Dr. Droste.M. *Recognizable languages in concurrency monoids.* Theoretical Computer Science 150 (1995), 77-109.

DK. Droste.M,Kuske.D. *Automata with concurrency relations, a survey.* Workshop on Algebraic and Syntactic Aspects of Concurrency.P.Gastin and A.Petit eds. LITP 95/48,1995.

DGP. Diekert.V,Gastin.P,Petit.A. *Rational and recognizable complex trace languages.* Information and Computation 116, 134-153, 1995.

F1. Fanchon.J. *A Fifo-net model for processes with asynchronous communication.* Lecture Notes in Computer Science, Vol. 609; Advances in Petri Nets 1992, 152-178. Springer-Verlag, 1992.
F2. Fanchon.J. *Trace Channel Nets.*LAAS report 98463,November 1998.
FB. Fanchon.J,Boyer.M. *Partial Order and Trace Channel Nets.*LAAS report 98158, May 1998.
FM. Finkel.A,Memmi.G. *Fifo Nets: A New Model of Parallel Computation.* Lecture Notes in Computer Science, Vol. 145: 111-121. Springer-Verlag, 1982.
GPZ. Gastin.P,Petit.A,Zielonka.W. *An extention of Kleene's and Ochmanski's theorems to infinite traces.* Theoretical Computer Science 125 (1991), 167-204.
Gi. Gischer. J. *The equational theory of pomsets.* Theoretical Computer Science 61 (1988) 199-224.
Gr. Grabowski.J. *On partial Languages.* Fundamenta Informaticae IV.2. (1981).
Je. Jensen.K. *Coloured Petri Nets.* Monographs in Theoretical Computer Science. Springer Verlag, 1995.
K1. Kiehn.A. *Comparing locality and causality based equivalences.* Acta Informatica 31, 697-718, 1994.
K2. Kiehn.A. *Observing Partial Order Runs of Petri Nets.* Lecture Notes in Computer Science, Vol 1337, Foundations of Computer Science, Springer -Verlag 1998.
LW. Lodaya.K, Weil.P. *Series-parallel pomsets:Algebra, Automata and Languages.* STACS 98, Lecture Notes in Computer Science, Vol. 1373, Springer -Verlag 1998.
Ma. Mazurkiewicz.A. *Basic notions of trace theory* .Lecture Notes in Computer Science, Vol. 354: 285-363. Springer Verlag, 1989.
NSW. Nielsen.M, Sassone.V, Winskel.G.*Relationships between models of concurrency.* Lecture Notes in Computer Science, Vol. 803, 425-476. Springer-Verlag 1993.
Oc. Ochmanski.E. *Regular behaviour of concurrent systems.* Bulletin of the European Association for Theoretical Computer Science n.27 (1985),56-67.
Pr. Pratt.W. *Modeling concurrency with partial orders.* Int.J.Parallel Programming 15 (1987) 33-71.
Vo. Vogler.W. *Modular Construction and Partial Order Semantics of Petri Nets.* Lecture Notes in Computer Science, Vol. 625. Springer-Verlag, 1992

Reasoning about Algebraic Generalisation of Petri Nets

Gabriel Juhás*

Technical University of Berlin
juhas@cs.tu-berlin.de

Abstract. In this paper we study properties of an abstract and (as we hope) a uniform frame for Petri net models, which enables us to generalise algebra as well as enabling rule of Petri nets. Our approach of such a frame is based on using partial groupoids in Petri nets. Properties of Petri nets constructed in this manner are investigated through related labelled transition systems. In particular, we investigate the relationships between properties of partial groupoids used in Petri nets and properties of transition systems crucial for the existence of the state equation and linear algebraic techniques. We show that partial groupoids embeddable into Abelian groups play an important role in preserving these properties.

1 Introduction

Petri nets are one of the first well established and widely used non-interleaving models of concurrent systems. Their origin arises from Carl Adam Petri's dissertation [27] and the later concept of vector addition systems [18]. Petri nets are popular and successfully used in many practical areas (see *e.g.* [14], Vol. III).

Briefly, a place/transition net (a p/t net), which is a basic version of Petri nets, is given by a set of places representing system components, a set of transitions representing a set of atomic actions of the system, and their relationship given by an input and output function associating with each transition a multi-set over the set of places [26,21]. A state of a net is a multi-set over the set of places called a marking. An occurrence of a transition removes/adds tokens from/to the marking according to the input/output function, respectively. A transition is enabled to occur iff in every place there are enough tokens to fire. Labelled transition systems (shortly transition systems) [19] represent the basic interleaving model of concurrency [31]. They may be described as directed graphs whose nodes are system states and whose arcs are labelled by elements from a set of labels representing a set of atomic actions of the system. For a Petri net,

* This work is a part of the joint research project "DFG-Forschergruppe Petrinetz-Technologie" between H. Weber (Coordinator), H. Ehrig (both from Technical University Berlin) and W. Reisig (Humboldt University Berlin) supported by the German Research Council (DFG). Part of this work was done during the author's stay at Institute of Control Theory and Robotics, Slovak Academy of Sciences, and BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation) Department of Computer Science, University of Aarhus.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 324–343, 1999.
© Springer-Verlag Berlin Heidelberg 1999

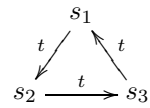
the related transition system, called sequential case graph, is the graph whose nodes are markings, and whose arcs are determined by possible occurrences of net transitions and labelled by these transitions.

A lot of effort has been put into the abstraction and extensions of Petri nets. In principle, we can recognise at least two kinds of Petri net extensions.

The first kind is formed by extensions which have the same expressiveness as p/t nets but enable us a more comprehensive description of systems - these extensions may be characterised as compressions of p/t nets. A typical example of this kind are nets commonly called high level nets [15], which introduce types of tokens. Another direction is adding hierarchy to Petri nets [10].

The second kind of extensions is characterised by an effort to have a more expressive model in comparison with p/t nets - these extensions might be called behavioural extensions of p/t nets. Again, behavioural extensions can be divided into at least two types. Extensions of the first type are based on relaxing the enabling rule. Here belong, among others, Petri nets with inhibitor arcs [26,13] that allow testing of zero markings, or contextual Petri nets [22], but also Controlled Petri nets [11], which are widely used in control of discrete event systems. The second type is characterised by generalising underlying algebra and extensions of this type are often characterised by an ambition to serve as an abstract and uniform frame for Petri net based models. Here we have to mention seminal work [21], some recent model of Abstract Petri nets introduced in [25], but also series of papers [5,4,30]. Another representants of extensions that use a different underlying algebra than p/t nets are extensions motivated by particular practical demands, *e.g* continuous and hybrid Petri nets [7,3].

Here, to give a simple and straightforward motivation of generalising underlying algebra of Petri nets, let us take the transition system $\mathcal{S}$ with three states $\{s_1, s_2, s_3\}$ and just one label, *i.e* just one atomic action t, given as follows:



Clearly, there does not exist any p/t net whose sequential case graph is (isomorphic to) the given transition system $\mathcal{S}$, because for an arbitrary set of places P all nonzero elements of the commutative monoid of multi-sets over P with multi-set addition have infinite order. In other words, the system $\mathcal{S}$ cannot be modelled by any p/t net with just one transition. To model this system by a Petri net, it is obvious to use a labelled Petri net[1], with at least two transitions labelled by the same label. However, using a cyclic group of order 3, such system would be modelled by an unlabelled Petri net with just one transition. Considering more complex systems with cyclic behaviour, *e.g.* transition systems with many cycles of different order, where each cycle is labelled using just one label, the number of necessary transitions in a labelled Petri net model may combinatorially grow with respect to the number of labels in the modelled system. To avoid this undesirable growth, it would suffice to use cyclic groups of the related orders.

[1] *i.e.* a Petri net whose transitions are labelled by elements of a set of labels [26]

Motivated by theoretical as well as practical demands, such as enabling an easy creation of new net models, or avoiding creation of similar concepts, techniques and results again and again for any particular extension, there is a real need to develop a sufficiently abstract and general frame that will capture the common spirit of Petri net models. However, the existing candidates for such a uniform frame of Petri net description, which suggest sufficiently abstract and general algebraic constructions, as for instance Abstract Petri nets from [25], do not deal with relaxing of enabling rule. Therefore, in [17] we have defined an abstract and uniform frame for various classes of Petri nets, which allows to generalise Petri net algebra as well as to work with a flexible enabling rule. Our approach of such a frame has been based on using partial algebra in Petri nets.

Given a class of Petri nets, the realisation problem, *i.e.* the question which kind of transition systems may be modelled by Petri nets from this class, has been studied deeply. There are exhaustive results for elementary net systems [8,24,12], the problem is solved for p/t nets in the setting of step transition systems in [23], and in special form also for Abstract Petri nets (including p/t nets) in [9,25], to mention only the most significant results. A very interesting recent framework, which characterises vector addition systems in a similar way as this paper, is [2].

Having a general frame that allows to define numerous Petri net classes, the dual question may be of interest. Namely, taking a property of transition systems, one may ask which class of Petri nets represents the class of transition systems with this property.

For illustration, the main advantages of p/t nets consist of the fact that they grant a well arranged graphical expression of the system as well as a simple analytical model in the form of a linear algebraic system called state equation which enables us to overcome sequentiality of computations. As a consequence, p/t nets offer linear algebraic analytical methods, such as place and transition invariants. The advantage of this simple analytical expression of p/t nets is due to following important properties of their sequential case graphs:

- The sequential case graphs of p/t nets are *deterministic* transition systems, *i.e.* transition systems, where for every state s and every label t there is at most one state s' such that there exists an arc from state s to state s' labelled by the label t;
- The sequential case graphs of p/t nets are *commutative* transition systems, *i.e.* transition systems, where for each pair of strings of labels with the same number of occurrences of single labels we have: if there exist paths from a common source state labelled by these strings then they lead to the common target state[2]. Commutativity is just the property that enables us to overcome sequentiality of computations, because it allows to use multi-sets of labels in a deterministic way.
- In addition, (the symmetric closures of) the sequential case graphs of p/t nets are *consensual* commutative transition systems, *i.e.* commutative transition systems, where for every two multi-sets of labels holds: if they change

[2] Hence, every commutative labelled transition system is deterministic.

a common source state to a common target state, then they change every common source state in which both may occur to a common target state. Moreover, this 'consensus' is transitive and closed under addition and subtraction of multi-sets. Thus, consensuality is just the property that enables us to speak about changes caused by multi-sets independently from states.

This may lead to an interesting question: Which properties of partial algebra used in extended Petri nets correspond to the properties of related transition systems crucial for the existence of the state equation and linear algebraic techniques, *i.e.* to the determinism, commutativity and consensuality of sequential case graphs? In [17] we have proved that the class of reachable consensual commutative transition systems coincides with the class of sequential case graphs of marked Petri nets whose underlying partial groupoid is embeddable into an Abelian group. However, one can easily find some non-reachable consensual commutative transition system for which there does not exist any Petri net with underlying algebra embeddable into an Abelian group whose sequential case graph is isomorphic with the given non-reachable transition system. So, in this paper we extend results from [17] proving that the class of sequential case graphs of unmarked Petri nets with partial groupoid embeddable into an Abelian group coincides with the class of transition systems whose *symmetric closure* is commutative and consensual.

2 Petri Nets - Basic Definitions

Before defining of Petri nets, let us first define some notation. We use $\mathbb{Z}$ to denote integers, $\mathbb{Z}^+$ to denote positive integers, and $\mathbb{N}$ to denote nonnegative integers. Moreover, we also shortly write $\mathbb{Z}$ to denote the infinite cyclic group of integers with addition $(\mathbb{Z}, +)$, and $\mathbb{N}$ to denote the commutative monoid of nonnegative integers with addition $(\mathbb{N}, +)$. Given two arbitrary sets, say A and B, symbol B^A denotes the set of all functions from A to B. Given a function f from A to B and a subset C of the set A we write $f|_C$ to denote the restriction of the function f on the set C. As usual, symbol 2^A denotes the power set (*i.e.* the set of all subsets) of the set A. To denote the set of all multi-sets over a set A with multi-set addition we write $(\mathbb{N}^A, +)$ or shortly $\mathbb{N}^A$. To denote the set of all finite multi-sets over a set A with multi-set addition, *i.e.* the free commutative monoid over a set A, we write $(\mathbb{N}^A_{fin}, +|_{\mathbb{N}^A_{fin} \times \mathbb{N}^A_{fin}})$. Thus, $\mathbb{N}^A_{fin} = \{b \mid b \in \mathbb{N}^A \wedge |A_b| \in \mathbb{N}\}$, where $A_b = \{a \mid a \in A \wedge b(a) \neq 0\}$ is a subset of the set A mapped by function $b \in \mathbb{N}^A$ to nonzero integers, *i.e.* A_b is the set of elements from A that are contained (occur at least once) in the multi-set b. Clearly, $(\mathbb{N}^A_{fin}, +|_{\mathbb{N}^A_{fin} \times \mathbb{N}^A_{fin}})$ is a submonoid of the monoid $(\mathbb{N}^A, +)$. Similarly, we use $(\mathbb{Z}^A, +)$ or shortly $\mathbb{Z}^A$ to denote the Abelian group of integer vectors over a set A with element by element addition. To denote the free Abelian group over a set A we write $(\mathbb{Z}^A_{fin}, +|_{\mathbb{Z}^A_{fin} \times \mathbb{Z}^A_{fin}})$. Thus, $\mathbb{Z}^A_{fin} = \{b \mid b \in \mathbb{Z}^A \wedge |A_b| \in \mathbb{N}\}$, where A_b is given as previously. Evidently, the free Abelian group $(\mathbb{Z}^A_{fin}, +|_{\mathbb{Z}^A_{fin} \times \mathbb{Z}^A_{fin}})$ is a subgroup of Abelian group $(\mathbb{Z}^A, +)$. As usual, we write only $(\mathbb{N}^A_{fin}, +)$ or shortly $\mathbb{N}^A_{fin}$ instead

of rather complicated $(\mathbb{N}^A_{fin}, +|_{\mathbb{N}^A_{fin}\times\mathbb{N}^A_{fin}})$; and $(\mathbb{Z}^A_{fin}, +)$ or shortly $\mathbb{Z}^A_{fin}$ instead of $(\mathbb{Z}^A_{fin}, +|_{\mathbb{Z}^A_{fin}\times\mathbb{Z}^A_{fin}})$. Notice that if the set A is finite, then $\mathbb{N}^A = \mathbb{N}^A_{fin}$ and $\mathbb{Z}^A = \mathbb{Z}^A_{fin}$. As one may see from the previous notation, we often use the symbol $+$ in the paper universally to denote a binary operation, *i.e.* we use the symbol $+$ for different operations.

According to the previous section, in algebraic form Petri nets (place/transition nets) [26,21] are defined as follows:

Definition 1. *A place/transition net (shortly p/t net) is an ordered tuple $\mathcal{N} = (P, T, I, O)$, where P and T are nonempty distinct sets of places and transitions, and $I, O : T \to \mathbb{N}^P$ are input and output functions. A marked p/t net is a pair $\mathcal{MN} = (\mathcal{N}, M_0)$, where $\mathcal{N}$ is a p/t net; and $M_0 \in \mathbb{N}^P$ is an initial marking.*

Definition 2. *A state of a p/t net called marking and denoted by M is a multiset over P,* i.e. *an element of $\mathbb{N}^P$. The dynamics of the net is expressed by the occurrence (firing) of enabled transitions. A transition $t \in T$ is enabled to occur in a marking $M \in \mathbb{N}^P$ iff $\forall p \in P : M(p) \geq I(t)(p)$,* i.e. *iff $\exists X \in \mathbb{N}^P : X + I(t) = M$. Occurrence of an enabled transition $t \in T$ in a marking M then leads to the new marking M' given by $M' = M + O(t) - I(t)$* i.e. *to $M' = X + O(t)$ for the $X \in \mathbb{N}^P$ such that $X + I(t) = M$.*

At this point we recall the definition of labelled transition systems [31]:

Definition 3. *A labelled transition system (shortly transition system) is an ordered tuple $\mathcal{S} = (S, L, \longrightarrow)$, where S is a set of states, L is a set of labels and $\longrightarrow \subseteq S \times L \times S$ is a transition relation.*

The fact that $(s, a, s') \in \longrightarrow$ is written as $s \xrightarrow{a} s'$. Denoting by L^* the monoid of all finite strings of labels from L with concatenation, it is obvious to extend the transition relation to *string transition relation* $\longrightarrow_\star \subseteq S \times L^* \times S$ as follows: $(s, q, s') \in \longrightarrow_\star$ whenever there exists a, possibly empty, string of labels $q = a_1 ... a_n$ such that $s \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s'$. To denote $(s, q, s') \in \longrightarrow_\star$ we simply write $s \xrightarrow{q}_\star s'$. Clearly, $s \xrightarrow{q}_\star s' \xrightarrow{v}_\star s'' \Longrightarrow s \xrightarrow{qv}_\star s''$. As usual, we also write $s \xrightarrow{a}$ and $s \xrightarrow{q}_\star$ to denote that there exists $s \in S$ such that $s \xrightarrow{a} s'$ and $s \xrightarrow{q}_\star s'$, respectively. A state s' is said to be reachable from a state s, iff there exists a string of labels q such that $s \xrightarrow{q}_\star s'$. Given a state $s \in S$, the set of all states reachable from s is denoted by $\{s \longrightarrow_\star\}$. A transition system is said to be reachable iff every $s' \in S$ is reachable from a fixed state $s \in S$ (*i.e.* $\exists s \in S : \{s \longrightarrow_\star\} = S$). A transition system $\mathcal{S} = (S, L, \longrightarrow)$ is called deterministic iff $\forall s \xrightarrow{a} s', s \xrightarrow{a} s'' : s' = s''$. As usual, we say that a state $s \in S$ is isolated iff there exists no $a \in L$ and no $s' \in S$ such that $s \xrightarrow{a} s' \vee s' \xrightarrow{a} s$.

We say that two transition systems are (transitionally) equivalent iff they differ only in some isolated states and unused labels, *i.e.* iff after removing some isolated states and unused labels they are isomorphic. Formally we have:

Definition 4. *Given any pair of transition systems $\mathcal{S} = (S, L, \longrightarrow)$ and $\mathcal{S}' = (S', L', \longrightarrow')$ we say that $\mathcal{S}$ and $\mathcal{S}'$ are (transitionally) equivalent iff there exists:*

– $S_1 \subseteq S, L_1 \subseteq L :\longrightarrow \subseteq S_1 \times L_1 \times S_1$
– $S_1' \subseteq S', L_1' \subseteq L' :\longrightarrow' \subseteq S_1' \times L_1' \times S_1'$

such that systems $(S_1, L_1, \longrightarrow)$ *and* $(S_1', L_1', \longrightarrow')$ *are isomorphic (*i.e. *there exist bijections* $\alpha : S_1 \to S_1'$ *and* $\beta : L_1 \to L_1'$ *such that* $\forall (s, a, s') \in S_1 \times L_1 \times S_1 :$ $s \xrightarrow{a} s' \iff \alpha(s) \xrightarrow{\beta(a)}{}' \alpha(s'))$.

Definition 5. *A pointed transition system is an ordered tuple* $\mathcal{PS} = (\mathcal{S}, i)$ *where* $\mathcal{S} = (S, L, \longrightarrow)$ *is a transition system; and* $i \in S$ *is a distinguished initial state such that* $\{i \longrightarrow_\star\} = S$, i.e. *every state is reachable from* i.

Now it is straightforward to see that each p/t net can be associated with a deterministic transition system.

Definition 6. *Let* $\mathcal{N} = (P, T, I, O)$ *be a p/t net. Then the transition system* $\mathcal{S} = (\mathbb{N}^P, T, \longrightarrow)$ *such that* $M \xrightarrow{t} M' \iff \Big(t$ *is enabled to occur in the marking* M *and its occurrence leads to the marking* $M'\Big)$ *is called sequential case graph of the p/t net* $\mathcal{N}$. *Given any marking* $M_0 \in \mathbb{N}^P$, *the pointed transition system* $\mathcal{PS} = (\{M_0 \longrightarrow_\star\}, T, \longrightarrow \cap \{M_0 \longrightarrow_\star\} \times T \times \{M_0 \longrightarrow_\star\}, M_0)$ *is called sequential case graph of the marked p/t net* $\mathcal{MN} = (\mathcal{N}, M_0)$.

In the following, for a finite string $q = a_1 \dots a_n$ over a set A we write b_q to denote Parikh's image of q, *i.e.* $b_q \in \mathbb{N}^A_{fin}$ is a multi-set in which the number of the occurrences $b_q(a)$ of each element a from A is given by the number of its occurrences in q, formally $b_q(a) = |\{i \mid i \in \{1, \dots, n\} \wedge a_i = a\}|$ for every $a \in A$.

Moreover, given a function $f : T \to \mathbb{Z}^P$, we denote by $\hat{f}$ the linear $\mathbb{Z}$-extension of the function f, *i.e.* we have $\hat{f} : \mathbb{Z}^T_{fin} \to \mathbb{Z}^P$ is such that $\forall b \in \mathbb{Z}^T_{fin} : \hat{f}(b) = \sum_{t \in T_b} f(t) \cdot b(t)$, where naturally the sum of empty set is zero-function, *i.e.* we define $\hat{f}(0) = 0$. In other words, for finite sets P and T, $\hat{f}$ may be understood as a 'matrix' whose rows are values $f(t)$, and then, for given 'vector' b, value $\hat{f}(b)$ represents the result of standard multiplication of the 'matrix' $\hat{f}$ by 'vector' b.

Now, given a p/t net $\mathcal{N} = (P, T, I, O)$, let $C : T \to \mathbb{Z}^P$ be such that $\forall t \in T :$ $C(t) = O(t) - I(t)$. Properties of the Abelian group $\mathbb{Z}^P$ over the set P enable us to write $M' = M + \hat{C}(b_q)$ whenever $M \xrightarrow{q}_\star M'$. Moreover, existence of a solution of the equation $\hat{C}(Y) = M' - M$ in $\mathbb{N}^T_{fin}$ is a necessary condition of reachability of M' from M. The solution $Y \in \mathbb{N}^T_{fin}$ then determines the number of transition occurrences that lead from M to M'. One usually calls the equation

$$M' = M + \hat{C}(Y) \qquad \text{or} \qquad M' = M + \hat{O}(Y) - \hat{I}(Y)$$

a *state equation* of p/t nets and $Y \in \mathbb{N}^T_{fin}$ a *firing vector*. Thus, in the p/t nets and their sequential case graphs the change of the state is invariant on the order of transition occurrences and depends only on the number of their occurrences[3].

In this place we give a definition of elementary nets adapted from [28,29] for which the realisation problem was already solved [8].

[3] However, the fact whether such a change of a state is possible depends on the order of transition occurrences.

Definition 7. *An elementary net is an ordered tuple* $\mathcal{EN} = (P, T, I, O)$ *where* P *and* T *are disjoint sets of places and transitions, and* $I, O : T \to 2^P$ *are input and output functions. A marking of an elementary net* $\mathcal{EN}$ *is a subset of* P*. A transition* $t \in T$ *is enabled to occur in a marking* $M \in 2^P$ *iff* $\exists X \subseteq P : X \cap I(t) = \emptyset \ \wedge \ X \cup I(t) = M \ \wedge \ X \cap O(t) = \emptyset$ *and then its occurrence leads to the marking* $M' = X \cup O(t)$*.*

3 Petri Nets - Algebraic Extensions

In this section we deal with some abstract generalisations of p/t nets found in the literature. One of the first successful abstractions of p/t nets can be found in the paper [21]. There the theory is built over Definition 1.

Recall that in [21] the possible occurrences of transitions are *represented* in a formally different way than we described in Definition 2. In fact, there is not explicitly defined enabling rule. In the following, let us briefly explain the way of representing the occurrences of transitions in the framework [21]:
First, recall that a directed graph is a quadruple (V, A, I, O), where V is a set of vertices, A is a set of arcs, and $I, O : A \to V$ are input (or source) and output (or target) functions, respectively. Thus, every p/t net $\mathcal{N} = (P, T, I, O)$ may be seen as a graph $\mathcal{G}_\mathcal{N} = (\mathbb{N}^P, T, I, O)$. Let $I', O' : \mathbb{N}^P \times \mathbb{N}^T_{fin} \to \mathbb{N}^P$ be such that

$$\forall (X, Y) \in \mathbb{N}^P \times \mathbb{N}^T_{fin} : I'(X, Y) = X + \hat{I}(Y) \ \wedge \ O'(X, Y) = X + \hat{O}(Y).$$

Then, directed graph $\mathcal{CG}_\mathcal{N} = (\mathbb{N}^P, \mathbb{N}^P \times \mathbb{N}^T_{fin}, I', O')$, which is just the reflexive and additive closure of graph $\mathcal{G}_\mathcal{N}$, is called case graph of the p/t net $\mathcal{N}$ in [21][4]. In graph $\mathcal{CG}_\mathcal{N}$, given an arc $(X, Y) \in \mathbb{N}^P \times \mathbb{N}^T_{fin}$, it represents a (possible) concurrent occurrence of transitions given by multi-set Y from the marking $M = I'(X, Y)$ to the marking $M' = O'(X, Y)$. Given a transition $t \in T$, all possible occurrences (firings) of single transition t are represented by the set of arcs $Arc_t = \{(X, Y) \mid (X, Y) \in \mathbb{N}^P \times \mathbb{N}^T_{fin} \ \wedge \ Y = b_t\}$[5]. So, in particular, given an arc $(X, Y) \in Arc_t$, it represents a possible occurrence (firing) of single transition t from the marking $M = I'(X, Y)$ to the marking $M' = O'(X, Y)$. Clearly, taking an arbitrary arc $(X, Y) \in Arc_t$, we have $I'(X, Y) = X + I(t)$ and $O'(X, Y) = X + O(t)$, and therefore, for all markings $M, M' \in \mathbb{N}^P$ and every transition $t \in T$, there exists a possible occurrence of single transition t from the marking M to the marking M' if and only if transition t is enabled to occur in the marking M and its occurrence leads to the marking M' according to Definition 2. This fact enables us to use Definition 2 whenever we refer to dynamics of nets from paper [21].

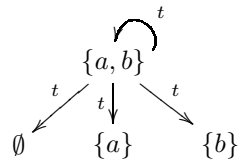
For our purpose the paper [21] is important by the fact that, to our best knowledge, it first proposes a generalisation of the algebra used for places of the

[4] Remember that case graph from [21], which is a directed graph, formally differs from the sequential case graph given in Definition 6, which is a *labelled* transition system.

[5] where b_t denotes the multi-set over set of transitions T determined by the one-element string t

net. As possible algebra there are suggested commutative monoids (the category of p/t nets with such assumption is called ***GralPetri*** in [21]), and generally semimodules. According to [21], for nets that are objects of ***GralPetri*** we have that markings are elements of a commutative monoid $(E, +)$, *i.e.* $M \in E$, and functions I, O associate with each transition an element of E, *i.e.* $I, O : T \to E$. Then, according to the previous paragraph, we have that a transition $t \in T$ is enabled to occur in a marking $M \in E$ iff there exists $X \in E$ such that $X + I(t) = M$, and then its occurrence leads to the marking M' such that $M' = X + O(t)$.

Example 1. Given a set $P = \{a, b\}$, take for the domain of markings non-cancellative commutative monoid formed by the power set of P with standard union, *i.e.* $(2^P, \cup)$. Let $T = \{t\}, I(t) = \{a, b\}$ and $O(t) = \emptyset$. Then we have that occurrence of t in marking $M = \{a, b\}$ leads to the marking M' such that $M' \cup \{a, b\} = M$. From the following figure with the sequential case graph of such net we can see that occurrence of t is nondeterministic.



The idea given in [21] is further generalised in [9] and [25]. The algebra considered is generally a commutative semigroup. So, according to [25], abstract Petri nets are given an follows:

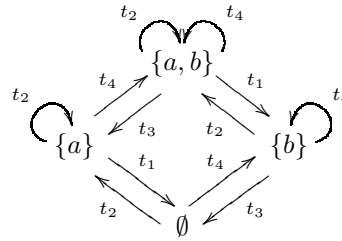
Definition 8. *A net structure functor is a composition $G \circ F$, where F is a functor from the category of sets to the category of commutative semigroups and G is its right adjoint. Given a net structure functor $G \circ F$, a low-level abstract Petri net is a tuple (P, T, I, O), where P and T are sets of places and transitions, respectively, and I, O are functions from T to $G \circ F(P)$.*

A marking of a low-level abstract Petri net is an element $M \in F(P)$. Because of the properties of adjunction, for each $f : T \to G \circ F(P)$ there exists a unique extension $\bar{f} : F(T) \to F(P)$[6]. Then the enabling rule and occurrence of "transition vectors" are defined as follows: given a "transition vector" $v \in F(\{t\})$, it is said that v is enabled to occur in M iff there exists $X \in F(P)$ such that $X + \bar{I}(v) = M$, and then the occurrence of v leads to $M' = X + \bar{O}(v)$. [25] mentions the possibility of nondeterminism caused by nonunique solutions of the equation $X + \bar{I}(v) = M$. It is solved by specifying additional conditions. Recall, that in [25] a net, where $F(P) = \mathcal{P}(P) = (2^P, \cup)$ is the standard powerset functor mapping a set to its power set with union, is called unsafe elementary net. The problem of nonunique solution of the equation $X \cup \bar{I}(v) = M$ is solved by demanding usual set complement of $\bar{I}(v)$ in M to be X.

[6] For more details see [25].

Example 2. Now, take again as in the previous example $P = \{a, b\}$, and an algebra of the net $(2^P, \cup)$. We have for a marking M that $M \in 2^P$. In order to remove nondeterminism of transition occurrences, let us consider only complements of $I(t)$ in M to be X in the equation $X \cup I(t) = M$, so we have that a transition t is enabled to occur in M iff there exists $X \in 2^P$ such that $X \cap I(t) = \emptyset \ \wedge \ X \cup I(t) = M$. Let $T = \{t_1, t_2, t_3, t_4\}$, let $I(t_1) = \{a\}, I(t_3) = \{b\}, O(t_2) = \{a\}, O(t_4) = \{b\}$ and let every other value of input and output function be equal to empty set[7]. So, we have that the occurrence of an enabled transition in a marking M leads to M' such that

$$\begin{array}{ll} \text{for } t_1 : M' \cup \{a\} = M; & \text{for } t_2 : M' = M \cup \{a\}; \\ \text{for } t_3 : M' \cup \{b\} = M; & \text{for } t_4 : M' = M \cup \{b\}; \end{array}$$



As we can see from the previous figure with the sequential case graph of the net, although demanding in the equation $X \cup I(t) = M$ only set complement of $I(t)$ in M to be X in order to remove nondeterminism, we have that, for example, the sequence of transitions $t_2t_1t_3$ changes the marking $\{a, b\}$ to the marking $\emptyset$, but the occurrence of the sequence of transitions $t_1t_3t_2$ changes the same marking $\{a, b\}$ to the marking $\{a\}$. This generally means that *using noncancellative commutative monoids as Petri net algebra, change of the state can depend on the order of the transition occurrences,* i.e. *commutativity of transition occurrences (if enabled) is not satisfied!* However, the commutativity is just the property that permits overcoming sequentiality in Petri net computations! There are also other interesting properties than commutativity preserved by partial algebras that can be embedded into an Abelian group, that holds in standard p/t nets but does not hold if one uses noncancellative commutative monoids. As we can see from the previous example, the occurrence of transition t_2 sometimes changes the marking (*e.g.* in $\{b\}$) and sometimes not (*e.g.* in $\{a\}$). These problems are caused by the fact, that in noncancellative commutative monoids equations can have more than one solution. However, if we add some other additional enabling conditions that choose just one solution, in Example 2 it could be the condition $X \cap O(t) = \emptyset$ that removes self-loops from the sequential case graph and makes from the net an elementary net established in Definition 7, then there are still some problems caused by the fact that the same functor is used for places and transitions. Namely, if one wants to allow the use of a composition of 'firing vectors' (*i.e.* to allow 'firing vectors' from $F(T)$), then the computation through sequences of 'firing vectors' may differ from the computation through the composition of 'firing vectors', *e.g.* in the case of Example 2 with previously mentioned additional

[7] Using the terminology from [25], we have defined an unsafe elementary net.

condition, the sequence of 'vectors' $\{t_1\}, \{t_3\}, \{t_2\}, \{t_1\}$ changes marking $\{a, b\}$ to the marking $\emptyset$, but composition $\{t_1\} \cup \{t_3\} \cup \{t_2\} \cup \{t_1\} = \{t_1, t_2, t_3\}$ changes the marking $\{a, b\}$ to the marking $\{a\}$. Generally, the use of the same functor in the construction of extended input and output functions $\bar{I}, \bar{O}$ causes that we cannot choose different kinds of algebra for transitions and places. However, for elementary net systems one could want to have a multi-set extension of input and output functions in order to express the change caused by multiple occurrences of transitions. Finally, the consideration of only total semigroups causes that the treatment of some more complicated enabling rule have to be done using some "additional conditions".

4 Algebraically Generalised Petri Nets, Transition Systems, and Abelian Groups

Based on the extensions discussed in the previous section, in this section we recall a very general algebraic structure of nets suggested in [17] for the purpose of investigating different Petri net extensions[8]. Then we choose a tuple of properties that holds in the sequential case graph of each standard p/t net. Further we show which kind of partial algebra, if it is used in p/t nets instead of integers with addition, preserves (is equivalent to) this tuple of properties.

As elementary nets[9] illustrate, it is often useful to have a more flexible enabling rule than that considered in standard p/t nets. Now the question is how to include the treatment of such more complicated enabling rule in a simple way into an abstract definition of Petri nets similar to that definition proposed in [21] and [25]. The use of partial groupoids solves this problem in an elegant way. In the following we recall very basic definitions and notation from theory of partial groupoids (according to [20]) used throughout the paper.

Definition 9. *A partial groupoid is an ordered tuple $\mathcal{H} = (H, \perp, \dot{+})$ where H is a carrier of $\mathcal{H}$, $\perp \subseteq H \times H$ is the domain of $\dot{+}$, and $\dot{+} : \perp \to H$ is a partial operation of $\mathcal{H}$.*

Definition 10. *We say that a partial groupoid $\mathcal{H} = (H, \perp, \dot{+})$ can be embedded (is embeddable) into an Abelian group iff there exists an Abelian group $(G, +)$ such that $H \subseteq G$ and the operation $+$ restricted on $\perp$ is equal to the partial operation $\dot{+}$, in symbols $+|_{\perp} = \dot{+}$. Group $(G, +)$ is called embedding of partial groupoid $\mathcal{H}$.*

Recall that a total groupoid is embeddable into an Abelian group if and only if it is a cancellative commutative semigroup. Remember that a left cancellative

[8] In this general definition we do not pay attention to distributivity of p/t nets, only to the algebra used. However, distributivity is a crucial property of p/t nets, it is included in 'end-user' definitions presented in [16].

[9] But also p/t nets with capacity and other classes of nets with more complicated enabling rule, such as Petri nets with inhibitor arcs [26,13], Contextual nets [22], or Controlled Petri nets [11].

semigroup is a semigroup where $\forall a, b, c \in H : a + b = a + c \implies b = c$. In similar way is defined right cancellative semigroup. A semigroup is said to be cancellative if it is both left and right cancellative. Clearly for a commutative semigroup left and right cancellativity coincides. For more details about embedding of semigroups into groups see *e.g.* [6]. In the case of proper partial groupoid (*i.e.* $\perp \subset H \times H$), associativity, commutativity and cancellativity are only necessary conditions of embeddability into an Abelian group, but not sufficient.

4.1 Algebraically Generalised p/t Nets

Let us denote the category of sets by **SET** and the category of partial groupoids by **PGROUPOID**. Further, let $U : \mathbf{PGROUPOID} \to \mathbf{SET}$ be the forgetful functor, *i.e.* given any partial groupoid $\mathcal{H} = (H, \perp, \dot{+})$, we have $U(\mathcal{H}) = H$.

Definition 11. *A net state functor is a functor $F : \mathbf{SET} \to \mathbf{PGROUPOID}$ associating a set with a partial groupoid. Given a net state functor F, an algebraically generalised place/transition net is an ordered tuple $\mathcal{AN} = (P, T, I, O)$, where P and T are distinct sets of places and transitions, and $I, O : T \to U \circ F(P)$ are input and output functions, respectively.*

Definition 12. *The structure $F(P)$ is called (partial) algebra of $\mathcal{AN}$. In the following, let $F(P) = (H, \perp, \dot{+})$. A state of net $\mathcal{AN}$, also called marking or case, is an element $M \in H = U \circ F(P)$. A transition $t \in T$ is enabled to occur in a state $M \in H$ if and only if $\exists X \in H$ such that $X \perp I(t) \wedge X \dot{+} I(t) = M \wedge X \perp O(t)$, and then its occurrence leads to the marking $M' = X \dot{+} O(t)$.*

Definition 13. *As usual, a marked algebraically generalised p/t net is a pair $\mathcal{MAN} = (\mathcal{AN}, M_0)$, where $\mathcal{AN}$ is an algebraically generalised p/t net and M_0 is a distinguished initial state of the net $\mathcal{AN}$.*

Definition 14. *Similarly to standard p/t nets, the sequential case graph of an algebraically generalised p/t net $\mathcal{AN} = (P, T, I, O)$ with partial algebra $F(P) = (H, \perp, \dot{+})$ is the transition system $(H, T, \longrightarrow)$, where $M \xrightarrow{t} M' \iff \Big(t$ is enabled to occur in the state M and its occurrence leads to the state $M'\Big)$,* i.e. *$M \xrightarrow{t} M' \iff \exists X \in H : X \perp I(t) \wedge X \dot{+} I(t) = M \wedge X \perp O(t) \wedge M' = X \dot{+} O(t)$. Then, given an initial state $M_0 \in H$, the sequential case graph of the marked algebraically generalised p/t net $\mathcal{MAN} = (\mathcal{AN}, M_0)$ is the pointed transition system $\mathcal{PS} = (\{M_0 \longrightarrow_\star\}, T, \longrightarrow \cap \{M_0 \longrightarrow_\star\} \times T \times \{M_0 \longrightarrow_\star\}, M_0)$.*

It is evident that standard p/t nets from Definition 1 are algebraically generalised nets with the state functor associating the set of places P with the monoid of all multi-sets over P with multi-set addition. Elementary nets from Definition 7 are algebraically generalised p/t nets with the state functor associating the set of places with the partial groupoid $(2^P, \perp, \uplus)$, where $\perp = \{(A, B) \mid A \cap B = \emptyset\}$ and $\uplus = \cup|_\perp$. Moreover, algebra of standard p/t nets, and partial algebra of elementary nets is embeddable into the Abelian group $\mathbb{Z}^P$.

4.2 Reasoning about Properties of Partial Algebra of Algebraically Generalised p/t Nets

In the following, we choose a tuple of properties, that holds in the sequential case graph of each standard p/t net. Then we find which kind of partial algebra used in algebraically generalised p/t nets preserves this tuple of properties.

Recall that given a finite string q over a set A we write b_q to denote Parikh's image of q, *i.e.* $b_q \in \mathbb{N}^A_{fin}$ is a multi-set in which the number of the occurrences $b_q(a)$ of each element a from A is given by the number of its occurrences in q.

Definition 15. *Let $\mathcal{S} = (S, L, \longrightarrow)$ be a transition system. We say that system $\mathcal{S}$ is commutative iff* $\forall\ s \xrightarrow{q}_\star s', s \xrightarrow{q'}_\star s'' : b_q = b_{q'} \Rightarrow s' = s''$.

Evidently, the sequential case graph of the net from Example 2 is not a commutative transition system. It is also clear that every commutative transition system is deterministic.

For commutative transition systems it is straightforward to extend the string transition relation $\longrightarrow_\star$ to *multi-set transition relation* $\longrightarrow_\diamond \subseteq S \times \mathbb{N}^L_{fin} \times S$ such that $(s, b, s') \in \longrightarrow_\diamond$ iff there exists $q \in L^*$ such that $s \xrightarrow{q}_\star s' \wedge b_q = b$. As usual, we write $s \xrightarrow{b}_\diamond s'$ to denote $(s, b, s') \in \longrightarrow_\diamond$ and $s \xrightarrow{b}_\diamond$ to denote that $\exists s' \in S : s \xrightarrow{b}_\diamond s'$. Clearly, $s \xrightarrow{b}_\diamond s' \xrightarrow{b'}_\diamond s'' \Longrightarrow s \xrightarrow{b+b'}_\diamond s''$.

Now recall that a congruence $\approx$ on an Abelian group $(G, +)$ is an equivalence relation on G preserving the group operation, *i.e.* $a \approx b\ \wedge\ c \approx d \Longrightarrow (a + c) \approx (b + d)$ for every $a, b, c, d \in G$. As usual, given an element $g \in G$, we denote by $[g]_\approx = \{g'\ |g' \approx g\}$ the equivalence class containing the element g, and by $+_{/\approx}$ the operation on $G/\approx\ = \{[g]_\approx | g \in G\}$ given as follows: $\forall [g]_\approx, [g']_\approx \in G/\approx :$ $[g]_\approx +_{/\approx} [g']_\approx = [g + g']_\approx$. In other words, $(G/\approx, +_{/\approx})$ is the factor group of the group $(G, +)$ according to the congruence $\approx$. For a more general statement of congruence we refer to [1].

Finally, let us define consensual commutative transition systems.

Definition 16. *Given a commutative transition system $\mathcal{S} = (S, L, \longrightarrow)$, let relation $\sim_\mathcal{S} \subseteq \mathbb{N}^L_{fin} \times \mathbb{N}^L_{fin}$ be such that $b \sim_\mathcal{S} b' \iff \exists\ s \xrightarrow{b}_\diamond s', s \xrightarrow{b'}_\diamond s'$. Let $\approx_\mathcal{S} \subseteq \mathbb{Z}^L_{fin} \times \mathbb{Z}^L_{fin}$ be the least congruence on $\mathbb{Z}^L_{fin}$ containing $\sim_\mathcal{S}$,* i.e. $\sim_\mathcal{S} \subseteq \approx_\mathcal{S}$. *We say that $\mathcal{S}$ is consensual iff* $\forall\ s \xrightarrow{b}_\diamond s', s \xrightarrow{b'}_\diamond s'' : b \approx_\mathcal{S} b' \Rightarrow s' = s''$.

In the following, we simply write $\sim$ and $\approx$ to denote $\sim_\mathcal{S}$ and $\approx_\mathcal{S}$ if system $\mathcal{S}$ is clear from the context. Also, given a consensual commutative transition system $\mathcal{S}$ we say that "$\mathcal{S}$ is commutative and consensual" instead of (maybe more precise but a bit laborious) "$\mathcal{S}$ is consensual commutative". We choose the name 'consensual' because it expresses a kind of 'common opinion' of the multi-sets on the problem "how to change the state". If they change a state in the same way once (from a common source state to a common target state) then they do it also for all other states in which they both can occur (if they are in any other common source state and both can occur, then they lead to a

common target state again). Moreover, this 'consensus' is transitive and closed under addition and subtraction of multi-sets.

Lemma 1. *The sequential case graph $(S, L, \longrightarrow)$ of every standard p/t net $\mathcal{N} = (P, T, I, O)$ (given by Def. 1) is commutative and consensual.*

Proof. For the sequential case graph of a p/t net $\mathcal{N}$ we have from its definition: $M \xrightarrow{q}_\star M' \Longrightarrow M' = M + \hat{C}(b_q)$ for every $M \xrightarrow{q}_\star M'$.

Commutativity: take any $M \xrightarrow{q}_\star M', M \xrightarrow{q'}_\star M''$ such that $b_q = b_{q'}$. From the previous result we have directly that $M' = M + \hat{C}(b_q) = M + \hat{C}(b_{q'}) = M''$. □

Now we show consensuality, *i.e.* we show that $\forall M \xrightarrow{b}_\diamond M', M \xrightarrow{b'}_\diamond M'' : b \approx b' \Rightarrow M' = M''$. Take $\cong \subseteq \mathbb{Z}^T_{fin} \times \mathbb{Z}^T_{fin}$ such that $b \cong b' \iff \hat{C}(b) = \hat{C}(b')$. One can easily check, that $\cong$ is a congruence on Abelian group $\mathbb{Z}^T_{fin}$ (clearly, it is an equivalence, and given any $\hat{C}(b) = \hat{C}(b')$ and $\hat{C}(d) = \hat{C}(d')$ we have $\hat{C}(b) + \hat{C}(b') = \hat{C}(d) + \hat{C}(d')$, and further $\hat{C}(b) + \hat{C}(b') = \sum_{t \in T} C(t) \cdot b(t) + \sum_{t \in T} C(t) \cdot b'(t) = \sum_{t \in T} C(t) \cdot (b(t) + b'(t)) = \hat{C}(b + b')$ and the same for d, d', *i.e.* we have $\hat{C}(b + b') = \hat{C}(d + d')$).

Given any $M \xrightarrow{b}_\diamond M', M \xrightarrow{b'}_\diamond M''$ we have that if $b \cong b$, (*i.e.* $\hat{C}(b) = \hat{C}(b')$ and therefore $M' = M + \hat{C}(b) = M + \hat{C}(b') = M''$) then $M' = M''$.
Because $\approx$ is the least congruence on $\mathbb{Z}^T_{fin}$ containing $\sim$, now it suffices to show that $\sim \subseteq \cong$. For every $b \sim b'$ we have from the definition of $\sim$ that $\exists M \xrightarrow{b}_\diamond M', M \xrightarrow{b'}_\diamond M'$, *i.e.* $M' = M + \hat{C}(b) = M + \hat{C}(b') \Longrightarrow \hat{C}(b) = \hat{C}(b')$, *i.e.* $b \cong b'$. □

Lemma 2. *Let $\mathcal{S} = (S, L, \longrightarrow)$ be a reachable transition system. Then $\mathcal{S}$ is commutative and consensual if and only if there exists an Abelian group $\mathcal{G} = (G, +)$ such that: $\exists$ an injection $\sigma : S \to G \ \wedge \ \exists f : L \to G$ such that $\forall\ s \xrightarrow{a} s' : \sigma(s) + f(a) = \sigma(s')$.*

Proof. $\Longrightarrow$ Because $\approx$ is a congruence on Abelian group $\mathbb{Z}^L_{fin}$, also $\mathcal{G} = (\mathbb{Z}^L_{fin}/\approx, +_{/\approx})$ is an Abelian group (it is a factor group of $\mathbb{Z}^L_{fin}$ with respect to $\approx$, so we have $\forall\ b, b' \in \mathbb{Z}^L_{fin} : [b + b']_\approx = [b]_\approx +_{/\approx} [b']_\approx$). System $\mathcal{S}$ is reachable, *i.e.* we have a state, say $r \in S$, from which each $s \in S$ is reachable. Now, let σ be defined as follows: $\sigma(r) = [0]_\approx$ and $\forall s \in S : \sigma(s) = [b]_\approx$ where $r \xrightarrow{b}_\diamond s$. One can check that σ is well defined (given any two b, b' such that $r \xrightarrow{b}_\diamond s$ as well as $r \xrightarrow{b'}_\diamond s$ there is $b \approx b'$ and therefore $[b]_\approx = [b']_\approx$), and it is an injection ($\sigma(s) = \sigma(s')$ means that there exists $r \xrightarrow{b}_\diamond s$ and $r \xrightarrow{b'}_\diamond s'$ such that $[b]_\approx = [b']_\approx$, *i.e.* $b \approx b'$, which, because $\mathcal{S}$ is consensual, implies $s = s'$).
Let $f : L \to \mathbb{Z}^L_{fin}/\approx$ be given by $f(a) = [b_a]_\approx$ for every $a \in L$.

Now take arbitrary $s \xrightarrow{a} s'$, *i.e.* $s \xrightarrow{b_a}_\diamond s'$. We have that $\sigma(s) = [b]_\approx$, where $r \xrightarrow{b}_\diamond s$. But then we also have $r \xrightarrow{b+b_a}_\diamond s'$ and therefore $\sigma(s') = [b + b_a]_\approx = [b]_\approx +_{/\approx} [b_a]_\approx = \sigma(s) +_{/\approx} f(a)$.
$\Longleftarrow$ as in Lemma 1. □

We have shown that in a reachable transition system commutativity and consensuality is equivalent to the existence of an Abelian group playing the role of algebra in the system. In the following a system in which computations may be expressed using elements and the composition law of an Abelian group is called Abelian group transition system.

Definition 17. *An Abelian group transition system is such a transition system* $\mathcal{S} = (S, L, \longrightarrow)$ *that there exists an Abelian group* $\mathcal{G} = (G, +)$ *such that:* $\exists$ *an injection* $\sigma : S \to G \ \wedge \ \exists f : L \to G$ *such that* $\forall s \xrightarrow{a} s' : \sigma(s) + f(a) = \sigma(s')$. *The group* $\mathcal{G} = (G, +)$ *is said to be associated with system* $\mathcal{S}$*, the injection* σ *is called state injection, and the function* f *is called incidence function of* $\mathcal{S}$*. If also* f *is an injection then it is said that* $\mathcal{S}$ *is unambiguously labelled.*

Corollary 1. *Clearly from Definition 1 of p/t nets, the sequential case graph of a standard p/t net* $\mathcal{N} = (P, T, I, O)$ *is an Abelian group transition system (it suffices to take* $\mathbb{Z}^P$ *as the associated group, inclusion from* $\mathbb{N}^P$ *to* $\mathbb{Z}^P$ *as the state injection, and function* C *such that* $\forall t \in T : C(t) = O(t) - I(t)$ *as the incidence function).*

For algebraically generalised p/t nets there hold the following claims:

Lemma 3. *The sequential case graph of an algebraically generalised p/t net* $\mathcal{AN} = (P, T, I, O)$ *in which partial algebra* $F(P)$ *can be embedded into an Abelian group, is an Abelian group transition system.*

Proof. Let $(G, +)$ denote an Abelian group into which the partial algebra $F(P)$ can be embedded . To show that the sequential case graph of $\mathcal{AN}$ is an Abelian group transition system it suffices to take $(G, +)$ as the associated group, inclusion from $F(P)$ to G as the state injection, and function $f : T \to G$ such that $\forall t \in T : f(t) = O(t) - I(t)$ as the incidence function. □

Lemma 4. *Every Abelian group transition system* $\mathcal{S} = (S, L, \longrightarrow)$ *is (transitionally) equivalent (see Def. 4) with the sequential case graph of an algebraically generalised p/t net* $\mathcal{AN}$ *in which partial algebra* $F(P)$ *can be embedded into an Abelian group.*

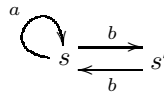
Proof. We have an Abelian group $(G, +)$ with a state injection $\sigma : S \to G$ and an incidence function $f : L \to G$ such that $\forall s \xrightarrow{a} s' : \sigma(s) + f(a) = \sigma(s')$. Without loss of generality, we can assume that $S \subseteq G \ \wedge \ \sigma(s) = s$ for every $s \in S$, and hence $\forall \ s \xrightarrow{a} s' : s + f(a) = s'$. Let $(K, +)$ be an Abelian group such that $L \subseteq K$. Now denote by $(H, +)$ direct product of $(G, +)$ and $(K, +)$ (*i.e.* $(H, +) = (G, +) \oplus (K, +)$, and we write (g, k) to denote an element of H, and 0 both for the neutral element of G and K). For every $a \in L$ let relation $\perp_a = (\{s \mid s \xrightarrow{a}\} \times \{-a\}) \times (\{f(a), 0\} \times \{a\}) \subseteq H \times H$. Finally denote $\perp = \cup_{a \in L} \perp_a$. So we have a partial algebra $\mathcal{H} = (H, \perp, +|_\perp)$ which can be embedded into Abelian group $(H, +)$. In the following we write as usual $\dot{+}$ instead of $+|_\perp$. Now take, for simplicity, the constant net state functor $F : \mathbf{SET} \to \mathbf{PGROUPOID}$ that

associates with each set partial groupoid $\mathcal{H}$ (and with each function between sets the identity morphism $id : \mathcal{H} \to \mathcal{H}$). Define an algebraically generalised net $\mathcal{AN} = (P, T, I, O)$, where P is an arbitrary set, $T = L$, and $I, O : T \to U \circ F(P)$ are given as follows:
$\forall a \in T : I(a) = (0, a) \wedge O(a) = (f(a), a)$
Let $(H, T, \rightarrowtail)$ be the sequential case graph of net $\mathcal{AN}$. From definition of $\perp$ (only $(s, -a) \perp (I(a) = (0, a))$ and $(s, -a) \perp (O(a) = (f(a), a))$ where $s \xrightarrow{a}$, are in the relation $\perp$) we have that $\forall\, a \in T\ \forall X \in H : X \perp I(a) \Longrightarrow (X \dot{+} I(a)) = (s, 0) \in S \times \{0\}$ and also $X \perp O(a) \Longrightarrow (X \dot{+} O(a)) = (s', 0) \in S \times \{0\}$. From the enabling and firing rule (recall that $a \in T$ is enabled in $M \in H$ iff there exists $X \in H : X \perp I(a) \ \wedge\ X \dot{+} I(a) = M \wedge X \perp O(a)$, and then occurrence (firing) of a leads to $M' = X \dot{+} O(a)$), we have that $\rightarrowtail \subseteq (S \times \{0\}) \times T \times (S \times \{0\})$, or in other words $\rightarrowtail\ = \longrightarrow \cap((S \times \{0\}) \times T \times (S \times \{0\}))$. Because of the construction of $\perp$ and the enabling and firing rule, we further have that

$$\forall a \in T\ \forall M \in S \times \{0\} : M = (s, 0) \overset{a}{\rightarrowtail} M' = (s', 0) \iff s \xrightarrow{a} s' \qquad (1)$$

So, there is an isomorphism (given by bijection $\beta : S \times \{0\} \to S,\ \beta(s, 0) = s$ for each $s \in S$; and identity $T = L$) between the transition system $\mathcal{S} = (S, L, \longrightarrow)$ and the transition system $(S \times \{0\}, T, \rightarrowtail)$. According to Definition 4, this immediately implies that system $\mathcal{S} = (S, L, \longrightarrow)$ is (transitionally) equivalent with the sequential case graph $(H, T, \rightarrowtail)$ of algebraically generalised p/t net $\mathcal{AN}$ with partial algebra $F(P)$ embeddable into Abelian group $(H, +)$. □

The importance of the group $(K, +)$ in the above proof is that it enables us to maintain the cases, where some transitions cause the same change but the domains on which they are enabled to occur differ. In particular, it allows to distinguish self-loops from each other and also from 'no' action in our construction. The consideration of possibly different domains for transitions with the same effect makes real sense in the case of ambiguous labelling of transition systems. We pay low price: we are still able to find an algebraically generalised p/t net with a partial algebra embeddable into an Abelian group, whose sequential case graph differs from the original system only in isolated states. This price is real in the sense that there exist Abelian group transition systems, for which a net whose partial algebra is embeddable into Abelian group and whose sequential case graph is isomorphic with the system, if we consider also isolated states, *does not exist*. For example, such a transition system is shown in the following figure.



It can be easily verified that for every algebraically generalised p/t net with $F(P)$ embeddable into an Abelian group, whose sequential case graph is equivalent with the previous system, the partial groupoid $F(P)$ must have more than two elements.

One may see the problem independently from Petri nets. Given a deterministic transition system $\mathcal{S} = (S, L, \longrightarrow)$ and an arbitrary Abelian group $(G, +)$

such that $S \subseteq G$, we can define a partial function $\delta : S \times L \to G$ defined for $(s, a) \in S \times L$ iff $s \xrightarrow{a}$, such that $s' = s + \delta(s, a)$ whenever $s \xrightarrow{a} s'$. Then easily, $\mathcal{S}$ is an Abelian group transition system iff there exists such an Abelian group that for the above mentioned function δ there holds:

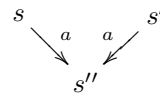
$$\forall a \in L \; \forall \; s, s' \in S : s \xrightarrow{a} \wedge \; s' \xrightarrow{a} \Longrightarrow \delta(s, a) = \delta(s', a). \tag{2}$$

Remark 1. If (2) holds, one can choose $f : L \to G$ such that $f(a) = \delta(s, a)$ where $s \in S$ is an arbitrary state such that $s \xrightarrow{a}$. On the other hand, if $\mathcal{S}$ is an Abelian group transition system one can choose $\delta(s, a) = f(a)$ for every $(s, a) \in S \times L$ such that $s \xrightarrow{a}$.

We should say that $\mathbb{Z}$-extension of a function, already mentioned in Section 2, can be treated more generally. For our purpose, given an Abelian group $(G, +)$ and a set A, the $\mathbb{Z}$-extension of a function $f : A \to G$ is the function $\hat{f} : \mathbb{Z}^A_{fin} \to G$ such that $\forall b \in \mathbb{Z}^A_{fin} : \hat{f}(b) = \sum_{a \in A_b} f(a) \cdot b(a)$, where the sum of empty set is zero-function, *i.e.* we have $\hat{f}(0) = 0$. Let us mention that previous construction is a special case of a semiring extension of a function from an arbitrary set to the carrier of a (left) semimodule over a semiring.

So, for an Abelian group transition system we have $\forall q \in L^* : s \xrightarrow{q}_\star s' \Longrightarrow s' = s + \hat{f}(b_q)$ whenever $s \xrightarrow{q}_\star s'$, or, in other words, we have that the existence of a nonnegative solution of the *state equation* $s' = s + \hat{f}(b_q)$ is a necessary condition for reachability of the state s' from the state s.

One may be interested in the problem *when a transition system $\mathcal{S}$ is an Abelian group transition system,* i.e. *what are the necessary and sufficient conditions for Abelian group transition systems.* According to Lemma 2, for reachable transition systems it is if and only if the system is commutative and consensual. The following figure gives a non-reachable commutative and consensual transition system, that is not an Abelian group transition system.



Now we would like to find similar properties to those in Lemma 2 also for transition systems which may not be reachable.

Given a transition system $\mathcal{S} = (S, L, \longrightarrow)$ let L^{-1} denote a set with the same cardinality as L, and $L \cap L^{-1} = \emptyset$. Let $\alpha : L \to L^{-1}$ be a bijection. Because the purpose of this is to define inverse elements of those in L, given $a \in L$ we denote $\alpha(a)$ by a^{-1}. We define $\longrightarrow^{-1} \subseteq S \times L^{-1} \times S$ as follows: $\longrightarrow^{-1} = \{(s', a^{-1}, s) \mid s \xrightarrow{a} s'\}$. Then, we define $\longmapsto \subseteq S \times (L \cup L^{-1}) \times S$ as the union of the relation $\longrightarrow$ and $\longrightarrow^{-1}$, *i.e.* $\longmapsto = \longrightarrow \cup \longrightarrow^{-1}$. Finally, the transition system $(S, L \cup L^{-1}, \longmapsto)$ is shortly denoted by $\mathcal{S}^{\leftrightarrow}$. In words, $\mathcal{S}^{\leftrightarrow}$ is a *symmetric closure* of $\mathcal{S}$ in which 'back-arrows' are denoted by 'inverse' labels. Directly from Definition 17 of Abelian group transition systems we have the corollary:

Corollary 2. *A transition system $\mathcal{S}$ is an Abelian group transition system if and only if its symmetric closure $\mathcal{S}^{\leftrightarrow}$ is an Abelian group transition system.*

Now we can formulate the following claim.

Lemma 5. *A transition system $\mathcal{S} = (S, L, \longrightarrow)$ is an Abelian group transition system if and only if its symmetric closure $\mathcal{S}^{\leftrightarrow}$ is commutative and consensual.*

Proof. $\Longrightarrow$ as in Lemma 1
$\Longleftarrow$ Denote by $\mathcal{C}$ a collection of equivalence classes of equivalence over relation $\smile \subseteq S \times S$ given by $s \smile s' \iff s \xrightarrow{a} s' \vee s' \xrightarrow{a} s$. Then a symmetric closure $\mathcal{S}^{\leftrightarrow}$ is a collection of isolated strongly connected components, *i.e.* mutually isolated reachable subsystems of $\mathcal{S}^{\leftrightarrow}$ with state sets given by equivalence classes from $\mathcal{C}$, in which $\forall C \in \mathcal{C}\ \forall s \in C : \{s \longmapsto_\star\} = C$. Now take a group, say $\mathcal{K} = (K, +, 0_K)$, with cardinality same as (or greater than) the collection $\mathcal{C}$ of these isolated strongly connected components, and an injection $\mu : \mathcal{C} \to K$. Take one element in every component, say s_C, for every component $C \in \mathcal{C}$. Take an injective function $\sigma : S \to \mathcal{K} \oplus (\mathbb{Z}_{fin}^{L \cup L^{-1}} / \approx_{\mathcal{S}^{\leftrightarrow}})$ which maps s_C to $(\mu(C), [0]_{\approx_{\mathcal{S}^{\leftrightarrow}}})$ for every $C \in \mathcal{C}$, and $\forall C \in \mathcal{C}\, \forall s \in C : \sigma(s) = (\mu(C), [b]_{\approx_{\mathcal{S}^{\leftrightarrow}}})$ where $s_C \xrightarrow{b}_\diamond s$. Looking in the proof of Lemma 2 we see that σ is well defined in S and injective inside of every component $C \in \mathcal{C}$, and because the states of different components are labelled by different elements of K we have that it is injective also in the whole set S. Now take $f : L \cup L^{-1} \to \{0_K\} \times \mathbb{Z}_{fin}^{L \cup L^{-1}} / \approx_{\mathcal{S}^{\leftrightarrow}}$ such that $\forall a \in L : f(a) = (0_K, [b_a]_{\approx_{\mathcal{S}^{\leftrightarrow}}})$. According to the fact that whenever $s \xrightarrow{a} s'$ we have that s and s' belongs to the same component, occurrences of transitions do not change K-component in the σ value of the new state, and following the way in the proof of Lemma 2, one can check that for every $s \xrightarrow{a} s'$ we have $\sigma(s)' = \sigma(s) + f(a)$. So, $\mathcal{S}^{\leftrightarrow}$ is an Abelian group transition system. According to the Corollary 2 also system $\mathcal{S}$ is an Abelian group transition system. □

Now we can complete our picture by showing which properties are equivalent to the use of a partial algebra embeddable into an Abelian group in algebraically generalised p/t nets also for sequential case graphs of unmarked p/t nets. Recall (from Corollaries 1 and 2) that the symmetric closure of the sequential case graph of each standard p/t net (Def. 1) is commutative and consensual. So, following Lemmas 3, 4 and 5 we formulate the theorem.

Theorem 1. *The symmetric closure of the sequential case graph of every algebraically generalised p/t net in which partial algebra $F(P)$ can be embedded into an Abelian group, is commutative and consensual. Every transition system whose symmetric closure is commutative and consensual is equivalent (with respect to Def. 4) to the sequential case graph of an algebraically generalised p/t net in which partial algebra $F(P)$ can be embedded into an Abelian group.*

We have shown properties corresponding to the use of a partial algebra embeddable into an Abelian group in Petri nets. One can choose a more general algebra, but should be prepared, that by doing this some of these properties can be lost (in the case of commutative monoids, as we demonstrated in Example 2,

we might even lose commutativity!). This is our main message. It is challenging to investigate the role of (partial) algebra used in Petri nets also through other properties than commutativity and consensuality of sequential case graphs.

We have also shown that every Abelian group transition system is transitionally equivalent with the sequential case graph of an algebraically generalised net whose partial algebra $F(P)$ is embeddable into an Abelian group. Following this result, one can investigate the relationship between transition systems and algebraically generalised p/t nets in more general settings: Given a class $\mathbf{A}$ of groupoids with some specified property, one can characterise the class $\mathbf{TS_A}$ of transition systems such that $(S, L, \longrightarrow) \in \mathbf{TS_A}$ iff $\exists (A, +) \in \mathbf{A}$ that $\exists$ an injection $\sigma : S \to A$ and $\exists\ f : L \to A$ such that $\forall s \xrightarrow{a} s' : \sigma(s) + f(a) = \sigma(s')$. We can investigate the relationship of such a class $\mathbf{TS_A}$ with algebraically generalised p/t nets, and in particular with algebraically generalised p/t nets the related (partial) algebra $F(P)$ of which is embeddable into a groupoid from class $\mathbf{A}$. For example, if we chose for class $\mathbf{A}$ the class of commutative monoids (so we have that $\mathbf{TS_A}$ is the class of commutative monoid transition systems) then evidently every system from $\mathbf{TS_A}$ is commutative. On the other hand, Example 2 shows that the sequential case graph $\mathcal{S}$ of the algebraically generalised p/t net, in which $F(P)$ is the commutative monoid of power set of places with union, is not a commutative transition system. In this general setting the equivalence of the class of Abelian group transition systems and the class of sequential case graphs of algebraically generalised p/t nets in which $F(P)$ is embeddable into an Abelian group (enabled by invertibility) seems to be specific.

5 Conclusion

In the paper we have suggested an extension of Petri nets based on generalising algebra as well as enabling rule used in the dynamics of nets. Our approach of such extension have been based on using partial groupoids in Petri nets. The motivation of the presented frame has been to offer an abstract and uniform frame for description of various behavioural extensions of Petri nets. Further, we have studied the role of partial algebra used in such algebraically generalised Petri nets through properties of their interleaving semantics - labelled transition systems called traditionally sequential case graphs. Our motivation and aim have been to determine the class of partial groupoids used in algebraically generalised Petri net which preserve some natural semantic properties guaranteed in standard Petri nets. We have started with a very general definition of Petri nets using a state functor mapping the set of places to partial groupoids, *i.e.* to Petri net partial algebra. For purpose of the study we have chosen the pair of semantic properties preserved in sequential case graphs of all standard Petri nets, namely *commutativity* of occurrences of transitions, which enables us to use multi-sets instead of sequences of transitions, and *consensuality* of computations in Petri nets (that means if the occurrences of two sequences of transitions in one state (marking) lead to the common new state, then they lead to the common new state from every other state in which both sequences are enabled to occur, and

moreover, this 'consensus' is transitive and closed under addition and subtraction of multi-sets). These two properties are crucial for the possibility of using linear algebraic techniques in the description and analysis of Petri nets. We have shown that using an arbitrary commutative monoid as Petri net algebra, as suggested in [21], does not preserve determinism and commutativity of occurrences of transitions in sequential case graphs of Petri nets, and although one uses some additional requirements to satisfy determinism, as it is done in [25], the sequential case graph still may not preserve commutativity. In other words, the construction of Petri nets causes that the commutativity of a monoid does not guarantee the commutativity of the corresponding sequential case graphs.

In this paper we have extended results from [17] also for non-reachable transition systems and unmarked Petri nets. Namely, we have proven that for a transition system $\mathcal{S}$ (not necessarily reachable) the following statements are equivalent:

- the symmetric closure of $\mathcal{S}$ is commutative and consensual;
- $\mathcal{S}$ is an Abelian group transition system, *i.e.* a system in which computations may be expressed using elements and the operation of an Abelian group;
- $\mathcal{S}$ is transitionally equivalent (*i.e.* differs only in isolated states) with the sequential case graph of an algebraically generalised Petri net the partial algebra of which is embeddable into an Abelian group.

Finally, let us mention that in [16] wide applicability of the presented frame has been illustrated in various examples.

Acknowledgement

I would like to thank Mogens Nielsen for valuable discussions, Zhe Yang for his extensive comments, technical suggestions and careful reading of the manuscript, Martin Drozda for reading and improvement of the draft of the paper, and Petr Jančar and Jörg Desel for their encouragement to continue on this work.

References

1. J. Adámek. *Theory of Mathematical Structures*. Kluwer, Dordrecht, 1983.
2. E. Badouel. Representation of Reversible Automata and State Graphs of Vector Addition Systems. Technical Report, INRIA Rennes, 1998.
3. J. Le Bail, H. Alla, and R. David. Hybrid Petri nets. In *Proceedings of 1st European Control Conference*, pp. 1472–1477, Grenoble, 1991.
4. P. Braun. Ein allgemeines Petri-netz. Master's thesis, Univ. of Frankfurt, 1992.
5. P. Braun, B. Brosowski, T. Fischer, and S. Helbig. A group theoretical approach to general Petri nets. Technical report, Univ. of Frankfurt, 1989-1991.
6. A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups I, II.* American Mathematical Society, 1961, 1967.
7. R. David and H. Alla. Continuous Petri nets. In *Proc. of 8th European Workshop on Application and Theory of Petri nets*, pp. 275–294, Zaragoza, 1987.
8. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures, part II: State spaces of concurrent systems. *Acta Informatika*, 27:343–368, 1990.

9. H. Ehrig, J. Padberg, and G. Rozenberg. Behaviour and realization construction for Petri nets based on free monoid and power set graphs. In *Workshop on Concurrency, Specification & Programming*. Humboldt University, 1994.
10. Xudong He. A formal definition of hierarchical predicate transition nets. In *Proc. ICATPN'96*, LNCS 1091, pp. 212–229, 1996.
11. L. E. Holloway, B. H. Krogh, and A. Giua. A survey of net methods for controlled discrete event systems. *Discrete Event Dynamic Systems*, 7:151–190, 1997.
12. H. J. Hoogeboom and G. Rozenberg. Diamond properties of elementary net systems. *Fundamenta Informaticae*, XIV:287–300, 1991.
13. R. Janicki and M. Koutny. Invariant semantics of nets with inhibitor arcs. In *Proc. CONCUR'91*, LNCS 527, 1991.
14. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use I II III*. Springer-Verlag, Berlin, 1992, 1995, 1997.
15. K. Jensen and G. Rozenberg, editors. *High-Level Petri-Nets, Theory and Applications*. Springer-Verlag, Berlin, 1991.
16. G. Juhás. *Algebraically Generalised Petri Nets*. PhD thesis, Institute of Control Theory and Robotics, Slovak Academy of Sciences, 1998.
17. G. Juhás. The essence of Petri nets and transition systems through Abelian groups. *Electronic Notes in Theoretical Computer Science*, 18, 1998.
18. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
19. R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):371–384, 1976.
20. E. S. Ljapin and A. E. Evseev. *Partial algebraic operations*. Izdatelstvo "Obrazovani", St. Petersburg, 1991. In Russian.
21. J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990.
22. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6):545–596, 1995.
23. M. Mukund. Petri nets and step transition systems. *International Journal of Foundations of Computer Science*, 3(4):443–478, December 1992.
24. M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.
25. J. Padberg. *Abstract Petri Nets: Uniform Approach and Rule-Based Refinement*. PhD thesis, Technical University of Berlin, 1996.
26. J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
27. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Univ. Bonn, 1962.
28. G. Rozenberg. Behaviour of elementary net systems. In *Proc. 2nd Advanced Course in Petri Nets, Advances in Petri Nets 1986*, LNCS 254, pp. 60–94,1987.
29. P. S. Thiagarajan. Elementary net systems. In *Proc. 2nd Advanced Course in Petri Nets, Advances in Petri Nets 1986*, LNCS 254, pp. 26–59, 1987.
30. P. Tix. One FIFO place realizes zero-testing and Turing machines. *Petri net Newsletter*, 51, 12 1996.
31. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pp. 1–148. Oxford University Press, 1995.

The Box Algebra – A Model of Nets and Process Expressions

Eike Best[1], Raymond Devillers[2], and Maciej Koutny[3]

[1] Fachb. Inf., Carl von Ossietzky Universität, D–26111 Oldenburg, Germany
[2] Départ. d'Inform., Université Libre de Bruxelles, B-1050 Bruxelles, Belgium
[3] Dept. of Comp. Sci., University of Newcastle, Newcastle upon Tyne NE1 7RU, U.K.

Abstract. The paper outlines a Petri net as well as a structural operational semantics for an algebra of process expressions. It specifically addresses this problem for the box algebra, a model of concurrent computation which combines Petri nets and standard process algebras. The paper proceeds in arguably the most general setting. For it allows infinite operators, and recursive definitions which can be unguarded and involve infinitely many recursion variables. The main result is that it is possible to obtain a framework where process expressions can be given two, entirely consistent, kinds of semantics, one based on Petri nets, the other on SOS rules.
Keywords: Net-based algebraic calculi; relationships between net theory and other approaches; process algebras; box algebra; refinement; recursion; SOS semantics.

1 Introduction

Concurrency theory is both challenging and difficult a subject of Computing Science such that, to this date, no single mathematical formalisation of it, of which there exist many, can claim to have absolute priority over others. This paper is about *combining* two widely known and well studied theories of concurrency: *process algebras* [1, 14, 15, 18] and *Petri nets* [2, 19, 22]. Process algebras: (i) allow the study of connectives directly related to actual programming languages; (ii) are compositional by definition; (iii) come with a variety of logics facilitating reasoning about important properties of systems; and (iv) support a variety of algebraic laws. On the other hand, Petri nets: (v) sharply distinguish between states and activities (the latter being defined as changes of state); (vi) treat global states and global activities as derived from their basic local counterparts; (vii) have a graphical representation which is easy to grasp and has therefore some wide appeal for practitioners; and (viii) have useful links both to graph theory and to linear algebra.

The work presented in this paper (in itself a continuation of, among others, [4, 5, 16]) does not subscribe to the ambition of trying to achieve a full combination of (i)–(viii) above – at least not immediately. Rather, it attempts to forge links between a fundamental (but restricted) class of Petri nets and a basic (but again restricted) process algebra. This paper will investigate the structural and

S.Donatelli, J.Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 344-363, 1999.
© Springer-Verlag Berlin Heidelberg 1999

behavioural aspects of these two basic models, and its main point is that there is, in fact, an extremely strong equivalence between them.

The *box algebra* is based on a set of process terms, called *box expressions*. Each box expression has associated two consistent kinds of semantics: a Petri net called a *box* (with its standard Petri net transition firing rule), and an operational semantics defined using SOS derivation rules [21]. A particular instance of the box algebra is the Petri Box Calculus (PBC) [3, 4, 9, 13] – a direct inspiration for the introduction of the box algebra. Note that the technique of associating Petri nets with process algebra expressions has also been studied for other models [7, 8, 11, 12, 20, 23].

The model we are going to describe is based on a set of operators, OpBox, which can be used to construct valid box expressions. For each operator $\mathsf{op} \in \mathsf{OpBox}$ there is an associated operator in the domain of boxes, Ω_{op}. This allows one to compositionally define, for every box expression $E = \mathsf{op}(E_1, E_2, \ldots)$, a corresponding net, $\mathsf{box}(E) = \Omega_{\mathsf{op}}(\mathsf{box}(E_1), \mathsf{box}(E_2), \ldots)$. The set of PBC operators includes sequence (denoted by the semicolon), choice (denoted by $\Box$), and parallel composition (denoted by $\|$). However, the box algebra supports a much richer set of constructs, including fully general recursion.

The two semantical models of the box algebra have been studied and developed in, e.g., [3, 5, 16]. In particular, it has been shown there that the two semantics are equivalent in the sense of generating strongly equivalent behaviours (in bisimulation sense [18]). This paper extends the already published results in three directions. First, the previous results were only applicable to finite operators; i.e., it was assumed that $\mathsf{op}(K_1, K_2, \ldots) = \mathsf{op}(K_1, K_2, \ldots, K_n)$, for some n. In this paper, op can in general take any number of arguments, in particular infinitely many. The interest in such general operators is motivated by a need to model operators like $\Box_{i \in I} E_i$, i.e., a fully generalised choice operator of Milner's CCS [18], which in turn can be used to give formal semantics to a process algebra with value-passing. Second, we remove two restrictions previously imposed on nets representing operators and operands; but we also analyse the conditions under which such a step does not compromise the behavioural integrity of the model. The third extension concerns consistency between the net semantics and operational semantics. We strengthen the previous results by stating that they generate, for every process expression, not only bisimulation equivalent, but in fact *isomorphic* transition systems; thus providing arguably the strongest possible consistency result.

The paper is organised as follows. In the next section, we introduce various classes of Petri nets used throughout the paper. Section 3 defines net refinement, the basic device to compose nets. Section 4 presents an algebra of process expressions based on the formalism developed in the preceding sections, and the last section states the above consistency result.

We use the standard mathematical notation. In particular, $\mathsf{mult}(Z)$ denotes the set of all finite multisets over a set Z, and $\uplus$ the disjoint set union. Other multiset-related notation are the sum (+) and difference (-).

2 Preliminaries

Each operator on nets will be based on a special net Ω_{op} whose transitions $v_1, v_2, \ldots$ are refined by the corresponding nets $\Sigma_1, \Sigma_2, \ldots$ in the process of forming a new net $\Omega_{\mathsf{op}}(\Sigma_1, \Sigma_2, \ldots)$. To carry this out, we need to distinguish nets that are easily composable with one another. The chosen class of Petri nets, called *boxes*, have interfaces expressed by labellings of places and transitions. There are two main classes of boxes which we will be interested in, viz. plain boxes and operator boxes. Plain boxes ($\Sigma_1, \Sigma_2, \ldots$ above) form the class of elements of our Petri net domain upon which various operators are defined. Operator boxes (Ω_{op} above) are patterns (or functions) defining the ways of constructing new plain boxes out of given ones.

Labelled Nets. We assume a set Lab of *actions* to be given; each $a \in \mathsf{Lab}$ represents some interface activity. A *relabelling* ρ is a relation $\rho \subseteq \mathsf{mult}(\mathsf{Lab}) \times \mathsf{Lab}$ such that $(\emptyset, a) \in \rho$ if and only if $\rho = \{(\emptyset, a)\}$. The intuition behind a pair (Γ, a) belonging to ρ is that it specifies some interface change which can be applied to a (finite) group of transitions whose labels match the argument, i.e., the multiset of actions Γ, and which are synchronised to yield a new transition labelled a.

A *constant* relabelling, $\rho_a = \{(\emptyset, a)\}$, where a is an action in Lab, can be identified with a itself, so that we may consider the set of actions Lab to be embedded in the set of all relabellings. If a relabelling is not constant, then it will be called *transformational*; in that case, the empty set will not be in its domain, in order not to *create an action out of nothing*. The *identity* relabelling, $\rho_{id} = \{(\{a\}, a) \mid a \in \mathsf{Lab}\}$ captures the 'keep things as they are' (non)change.

By a *(marked) labelled net* we will mean a tuple $\Sigma = (S, T, W, \lambda, M)$ such that: S and T are disjoint sets of respectively *places* and *transitions*; W is a *weight function* from the set $(S \times T) \cup (T \times S)$ to the set of natural numbers $\mathbf{N}$; λ is a *labelling* function for places and transitions such that $\lambda(s) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$, for every place $s \in S$, and $\lambda(t)$ is a relabelling ρ, for every transition $t \in T$; and M is a *marking*, i.e., a mapping assigning a natural number to each place $s \in S$.[1] We adopt the standard rules about representing nets as directed graphs. To avoid ambiguity, we will sometime decorate the various components of Σ with the index Σ; thus, T_Σ denotes the set of transitions of Σ, etc. A net is *finite* if both S and T are finite sets. Figure 1 shows the graph of a labelled net Σ_0.

If the labelling of a place s is e, i or x, then s is an *entry*, *internal* or *exit* place, respectively. By convention, ${}^\circ\Sigma$, Σ° and $\ddot{\Sigma}$ denote respectively the entry, exit and internal places of Σ. For every place (transition) x, we use ${}^\bullet x$ to denote is pre-set, i.e., the set of all transitions (places) y such that there is an arc from y to x, that is, $W(y, x) > 0$. The post-set $x^\bullet$ is defined in a similar way. The pre- and post-set notation extends in the usual way to sets R of places and transitions, e.g., ${}^\bullet R = \bigcup_{r \in R} {}^\bullet r$. In what follows, all nets are assumed to be T-*restricted*, i.e., the pre- and post-sets of each transition are nonempty. Σ is called *simple* if W always returns 0 or 1, and *pure* if for all transitions $t \in T$, ${}^\bullet t \cap t^\bullet = \emptyset$.

[1] Sometimes we will treat M as a set, if $M(S) \subseteq \{0, 1\}$.

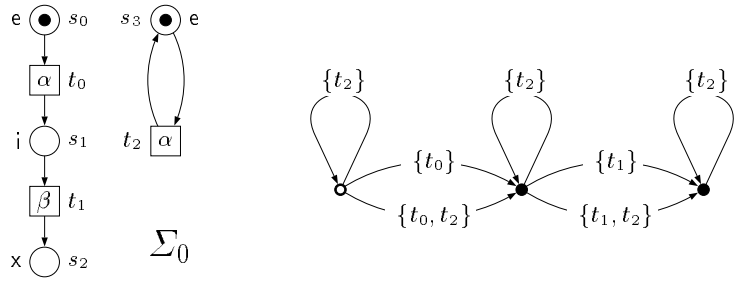


Fig. 1. A labelled net and its reachability graph.

For the labelled net of figure 1 we have ${}^\circ\Sigma_0 = \{s_0, s_3\}$, $\Sigma_0^\circ = \{s_2\}$, ${}^\bullet s_0 = \emptyset$ and $\{s_0, s_1\}^\bullet = \{t_0, t_1\}$. This net is finite and simple, but not pure as $s_3 \in {}^\bullet t_2 \cap t_2^\bullet$.

We will use three explicit ways of modifying the marking of Σ. We define $\lfloor \Sigma \rfloor$ as $(S, T, W, \lambda, \emptyset)$ which amounts to erasing all the tokens. Moreover, $\overline{\Sigma}$ and $\underline{\Sigma}$ are, respectively, $(S, T, W, \lambda, {}^\circ\Sigma)$ and $(S, T, W, \lambda, \Sigma^\circ)$. These operations correspond to placing one token on each entry (resp. exit) place and nothing elsewhere, thus constituting the *entry marking* (resp. the *exit marking*).

Execution Semantics. The behaviour of Σ is defined by its finite step sequence semantics: a finite multiset of transitions U, called a *step*, is *enabled* by Σ if for every place $s \in S$, $M(s) \geq \sum_{t \in U} (U(t) \cdot W(s,t))$. We denote this by $M\ [U\rangle$. An enabled step U can be *executed* leading to a follower marking M' defined, for every $s \in S$, by $M'(s) = M(s) + \sum_{t \in U} (U(t) \cdot (W(t,s) - W(s,t)))$. We will denote this by $\Sigma\ [U\rangle\ \Sigma'$, where $\Sigma' = (S, T, W, \lambda, M')$. For Σ_0 in figure 1, $\{t_0, t_2\}$ is an enabled step. After its execution, $\{t_1\}$ is enabled and, hence, $\{t_0, t_2\}\{t_1\}$ is a step sequence of Σ_0. Formally, a *step sequence* of Σ is a possibly empty sequence of steps, $\omega = U_1 \ldots U_k$, such that there are nets $\Sigma_1, \ldots, \Sigma_k$ satisfying $\Sigma\ [U_1\rangle\ \Sigma_1\ [U_2\rangle\ \cdots\ [U_k\rangle\ \Sigma_k$. We will denote this by $\Sigma\ [\omega\rangle\ \Sigma_k$ or $\Sigma_k \in [\Sigma\rangle$.

A marking M is *safe* if $M(S) \subseteq \{0, 1\}$. A safe marking is *clean* if it is not a proper superset of ${}^\circ\Sigma$ (the *entry* marking) nor Σ° (the *exit* marking), i.e., if ${}^\circ\Sigma \subseteq M$ or $\Sigma^\circ \subseteq M$ implies ${}^\circ\Sigma = M$ or $\Sigma^\circ = M$, respectively. The marking of the net in figure 1 is both safe and clean. A labelled net Σ is: *ex-restricted* if ${}^\circ\Sigma \neq \emptyset \neq \Sigma^\circ$; *e-directed* if ${}^\bullet({}^\circ\Sigma) = \emptyset$; *x-directed* if $(\Sigma^\circ)^\bullet = \emptyset$; and *ex-directed* if it is both e-directed and x-directed.

Behavioural Equivalence. Although the whole set of step sequences of a net may be specified by defining the *full reachability graph* (see [22] and figure 1), we do not find it a satisfactory representation in the presence of labellings. For example, figure 2 demonstrates that isomorphism (and, indeed, other reasonable notion of behavioural equivalence) of reachability graphs is not preserved by sequential composition of nets. This is due to the fact that it is necessary to distinguish the entry and exit markings when comparing the behaviour of nets which are subsequently composed. Instead of modifying the definition of isomorphism, we address this problem by adding to Σ (artificially) two fresh transitions, skip and redo, so that ${}^\bullet\mathsf{skip} = \mathsf{redo}^\bullet = {}^\circ\Sigma$, $\mathsf{skip}^\bullet = {}^\bullet\mathsf{redo} = \Sigma^\circ$, $\lambda(\mathsf{skip}) = \mathsf{skip}$

and $\lambda(\mathsf{redo}) = \mathsf{redo}$. Moreover, all the arcs adjacent to the skip and redo transitions are unitary and $\mathsf{redo}, \mathsf{skip} \notin \mathsf{Lab}$. Denote the net Σ augmented with skip and redo by Σ_{sr}. Then the *transition system* generated by Σ with $M_\Sigma \neq \emptyset$ is defined as $\mathsf{ts}_\Sigma = (V, L, A, v_0)$ where $V = \{\Theta \mid \Theta_{\mathsf{sr}} \in [\Sigma_{\mathsf{sr}}\rangle\}$ is the set of states, $v_0 = \Sigma$ is the initial state, $L = \mathsf{mult}(T_\Sigma \uplus \{\mathsf{redo}, \mathsf{skip}\})$ is the set of arc labels, and the arcs are given by: $A = \{(\Theta, U, \Psi) \mid \Theta_{\mathsf{sr}} \in [\Sigma_{\mathsf{sr}}\rangle \wedge \Theta_{\mathsf{sr}}\,[U\rangle\,\Psi_{\mathsf{sr}}\}$ (see also figure 2). In other words, ts_Σ is the reachability graph of Σ_{sr} with all the references to skip and redo at the nodes (but not arcs) of the graph erased. The transition system generated by Σ with $M_\Sigma = \emptyset$ is defined as $\mathsf{ts}_\Sigma = \mathsf{ts}_{\overline{\Sigma}}$. Figure 2 shows that adding skip and redo does solve the problem; although the nets Σ and Θ have isomorphic reachability graphs, their ts's are different. This is not a mere chance; ts-isomorphism turns out to be a *congruence* in the net algebra defined in the next section. It therefore follows that ts_Σ can be accepted as a good representation of the global behaviour of Σ.

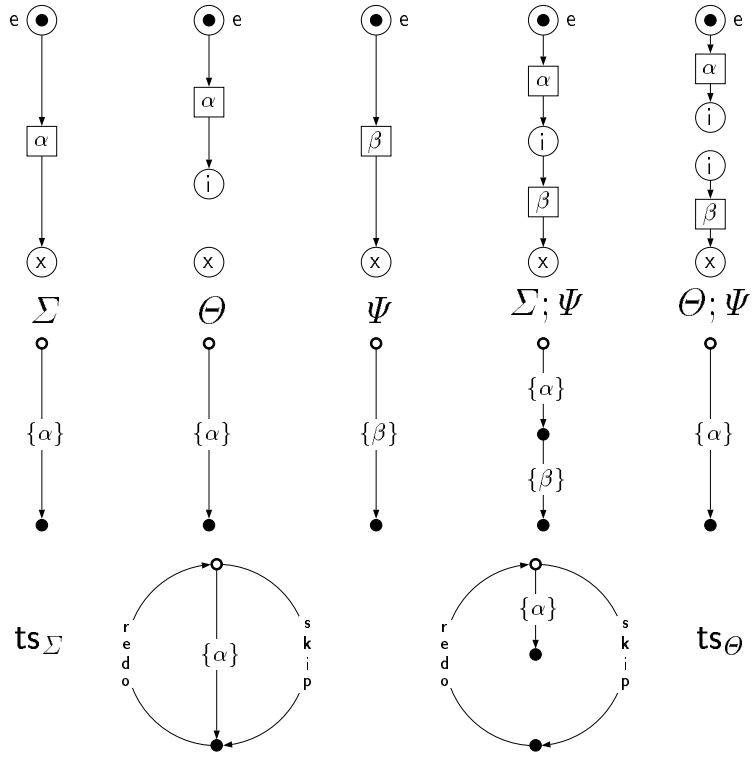


Fig. 2. Five nets and the corresponding (labelled) full reachability graphs demonstrating that isomorphism of reachability graphs is not preserved by sequential composition; the two discriminating ts's for Σ and Θ are also shown.

Plain Boxes. A *box* is a (possibly infinite) ex-restricted labelled net Σ. It is, by definition, *plain* if for each transition $t \in T_\Sigma$, the label $\lambda_\Sigma(t)$ is a constant

relabelling, i.e., $\lambda_\Sigma(t) \in \mathsf{Lab}$. A plain box Σ is *static* if $M_\Sigma = \emptyset$ and every marking reachable from ${}^\circ\Sigma$ or Σ° is safe and clean. A plain box Σ is *dynamic* if $M_\Sigma \neq \emptyset$ and every marking reachable from M_Σ or ${}^\circ\Sigma$ or Σ° is safe and clean. A dynamic box Σ is an *entry* (*exit*) box if $M_\Sigma = {}^\circ\Sigma$ (resp. $M_\Sigma = \Sigma^\circ$). Note that the labelled net Σ_0 in figure 1 is not a box since it has a reachable marking which is not clean. The sets of plain static, dynamic, entry and exit boxes will, respectively, be denoted by Box^s, Box^d, Box^e and Box^x.

Operator Boxes. To define an operator box we take a simple box Ω with all relabellings being transformational such that $M_\Omega = \emptyset$ and all the markings reachable from ${}^\circ\Omega$ or Ω° are safe and clean. But we still need to impose on Ω one more property, called *factorisability* [16], defined next.

As the transitions of Ω are meant to be refined by potentially complicated boxes, it seems reasonable to consider that their execution may take long time or, indeed, may last indefinitely. This may be captured by a special kind of extended markings.

A *complex marking* of Ω is a pair (M, Q) composed of a normal marking M of Ω and a finite multiset Q of *engaged* transitions of Ω. A standard marking M may then be identified with the complex marking $(M, \emptyset)$. The *direct reachability* between two complex markings (M, Q) and (M', Q') is defined thus. We have $(M, Q) \to (M', Q')$ if there are finite multisets of transitions U, V and Z such that $Z \subseteq Q$, $Q' = Q + V - Z$ and for every $s \in S$, $M(s) \geq m_s$ and $M'(s) = M(s) - m_s + m'_s$, where $m_s = \sum_{t \in U+V} W(s,t) \cdot (U(t) + V(t))$ and $m'_s = \sum_{t \in U+Z} W(t,s) \cdot (U(t) + Z(t))$.

A tuple of sets of transitions of Ω, $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ is a *factorisation* of a complex safe (i.e., both M and Q are sets) marking (M, Q) if $\mu_d = Q$, $\mu_s = T_\Omega \setminus (\mu_e \cup \mu_d \cup \mu_x)$ and $M = \biguplus_{v \in \mu_e} {}^\bullet v \uplus \biguplus_{v \in \mu_x} v^\bullet$. Ω itself will be called *factorisable* if for every safe complex marking of Ω reachable from the entry marking of Ω_{sr}, there is at least one factorisation. We will denote by fact_Ω the set of all the factorisations of all the complex markings of Ω reachable from the entry marking, including the only factorisation $(\emptyset, \emptyset, \emptyset, T_\Omega)$ of the empty marking.

The notion of an operator box we have just defined corresponds to what is called an *sos-operator box* in [5, 16] where the reasons for requiring factorisation of Ω are explained in full detail.

Let $\mathbf{\Sigma} : T_\Omega \to \mathsf{Box}^s \cup \mathsf{Box}^d$ be a function from the transitions of Ω to static and dynamic boxes. We will refer to $\mathbf{\Sigma}$ as an *Ω-tuple* and denote $\mathbf{\Sigma}(v)$ by Σ_v, for every $v \in T_\Omega$. If the set of transitions of Ω is finite, we assume that $T_\Omega = \{v_1, \ldots, v_n\}$ and then denote $\mathbf{\Sigma} = (\Sigma_{v_1}, \ldots, \Sigma_{v_n})$. We extend the notion of factorisation to an Ω-tuple $\mathbf{\Sigma}$ of static and dynamic boxes; the *factorisation* of $\mathbf{\Sigma}$ is the quadruple $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ such that $\mu_\delta = \{v \mid \Sigma_v \in \mathsf{Box}^\delta\}$ for $\delta \in \{e, x, s\}$, and $\mu_d = \{v \mid \Sigma_v \in \mathsf{Box}^d \setminus (\mathsf{Box}^e \cup \mathsf{Box}^x)\}$.

The *domain of application* of Ω, denoted by dom_Ω, is then defined as the set comprising every Ω–tuple of static and dynamic boxes $\mathbf{\Sigma}$ whose factorisation belongs to fact_Ω, and such that, for every $v \in T_\Omega$:

Dom1 If ${}^\bullet v \cap v^\bullet \neq \emptyset$ then Σ_v is *ex-exclusive*, where the latter means that, for every marking M reachable from M_Σ or ${}^\circ\Sigma$ or Σ°, it is the case that $M \cap {}^\circ\Sigma = \emptyset$ or $M \cap \Sigma^\circ = \emptyset$.
Dom2 If v is not reversible then Σ_v is x-directed, where v is *reversible* if, for every complex marking (M, Q) reachable from ${}^\circ\Omega$ or Ω°, $v^\bullet \subseteq M$ implies that $(M \setminus v^\bullet, Q \cup \{v\})$ is also a marking reachable from ${}^\circ\Omega$ or Ω°.

In the previously published work on the box algebra, it was assumed that all boxes are ex-directed and, in addition, that operator boxes are both finite (though this restriction was not used in [10]) and pure. Making such assumptions resulted in a simplification of the formal treatment and proofs. The present framework is more difficult to handle, but at the same time it is much more expressive with obvious implications for the practical applicability of the box algebra.

3 Net Refinement

Net refinement embodies a mechanism by which transition refinement and interface change specified by relabellings are combined. Both operations are defined for an operator box Ω which serves as a pattern for gluing together a tuple of plain boxes $\mathbf{\Sigma}$ along their entry and exit interfaces. The relabellings annotating the transitions of Ω specify the interface changes to which the boxes in $\mathbf{\Sigma}$ are subjected.

As far as net refinement as such is concerned, the names (identities) of newly constructed transitions and places are basically irrelevant. However, in our approach to solving recursive definitions on boxes, the *names* of places and transitions play a crucial role since they are a key in the construction of recursive nets [3, 10, 16].

We shall assume that there are two disjoint infinite sets of *basic* place and transition names, $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$. Each name $\eta \in \mathsf{P}_{\mathsf{root}} \cup \mathsf{T}_{\mathsf{root}}$ can be viewed as a special tree with a single root labelled with η which is also a leaf. We shall also employ more complex trees as transition and place names, and use a linear notation to express such trees. To this end, an expression $x \lhd \mathsf{S}$, where x is a basic name in $\mathsf{P}_{\mathsf{root}} \cup \mathsf{T}_{\mathsf{root}}$ or a pair $(t, a) \in \mathsf{T}_{\mathsf{root}} \times \mathsf{Lab}$, and S is a multiset of trees, denotes a tree where the trees of the multiset are appended (with their multiplicity) to an x-labelled root. Moreover, if $\mathsf{S} = \{p\}$ is a singleton then $x \lhd \mathsf{S}$ will be denoted by $x \lhd p$, and if S is empty then $x \lhd \mathsf{S} = x$.

We shall further assume that in every operator box, all the places and transitions are basic names (i.e., single root trees) from respectively $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$. For the plain boxes, the trees used as names may be more complex. Each transition tree is a finite tree labelled with elements of $\mathsf{T}_{\mathsf{root}}$ (at the leaves) and $\mathsf{T}_{\mathsf{root}} \times \mathsf{Lab}$ (elsewhere), and each place tree is a possibly infinite (in depth and width) tree labelled with basic names from $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$, which has the form $t_1 \lhd t_2 \lhd \ldots \lhd t_n \lhd s \lhd \mathsf{S}$, where $t_1, \ldots, t_n \in \mathsf{T}_{\mathsf{root}}$ $(n \geq 0)$ are transition names and $s \in \mathsf{P}_{\mathsf{root}}$ is a place name (so that no confusion will be possible between transition-trees and place-trees: the latter always have a label from $\mathsf{P}_{\mathsf{root}}$ and the

former never). We comprise all these trees (including the basic ones consisting only of a root as special cases) in our sets of allowed transition and place names, denoted respectively by $\mathsf{P}_{\mathsf{tree}}$ and $\mathsf{T}_{\mathsf{tree}}$.

Formal Definition. Let Ω be an operator box, and $\Sigma \in \mathsf{dom}_\Omega$ be an Ω-tuple of static and dynamic boxes in its domain of application. The result of a simultaneous substitution of the boxes Σ for the transitions in Ω is a labelled net $\Omega(\Sigma)$ such that the set of places is defined as the (disjoint) union $S_{\Omega(\Sigma)} = \bigcup_{v \in T_\Omega} \mathsf{ST}^v_{\mathsf{new}} \cup \bigcup_{s \in S_\Omega} \mathsf{SP}^s_{\mathsf{new}}$ where $\mathsf{ST}^v_{\mathsf{new}} = \{\, v \lhd q \mid q \in \ddot{\Sigma}_v \}$ and $\mathsf{SP}^s_{\mathsf{new}}$ is the set of all

$$p \;=\; s \lhd (\{v \lhd x_v\}_{v \in {}^\bullet s} + \{w \lhd e_w\}_{w \in s^\bullet}) \tag{1}$$

such that $x_v \in \Sigma_v^\circ$ and $e_w \in {}^\circ\Sigma_w$. The marking (label) of a place p in $\Omega(\Sigma)$ is: $M_{\Sigma_v}(q)$ (resp. i) if $p = v \lhd q \in \mathsf{ST}^v_{\mathsf{new}}$, and

$$\sum\nolimits_{v \in {}^\bullet s} M_{\Sigma_v}(x_v) + \sum\nolimits_{w \in s^\bullet} M_{\Sigma_w}(e_w)$$

(resp. $\lambda_\Omega(s)$) if $p \in \mathsf{SP}^s_{\mathsf{new}}$ is as in (1). For a transition y of Ω and a place $p = v \lhd q \in \mathsf{ST}^v_{\mathsf{new}}$, we define $\mathsf{trees}^y(p) = \{q\}$ if $v = y$, and $\mathsf{trees}^y(p) = \emptyset$ if $v \neq y$; moreover, for $p \in \mathsf{SP}^s_{\mathsf{new}}$ as in (1), we define $\mathsf{trees}^y(p)$ as: $\{x_y, e_y\}$ if $y \in {}^\bullet s \cap s^\bullet$; $\{x_y\}$ if $y \in {}^\bullet s \setminus s^\bullet$; $\{e_y\}$ if $y \in s^\bullet \setminus {}^\bullet s$; and $\emptyset$ otherwise.

The set of transitions is $T_{\Omega(\Sigma)} = \bigcup_{v \in T_\Omega} \mathsf{T}^v_{\mathsf{new}}$ where $\mathsf{T}^v_{\mathsf{new}}$ comprises all $t = (v, a) \lhd Q$ such that $Q \in \mathsf{mult}(T_{\Sigma_v})$ and $(\lambda_{\Sigma_v}(Q), a) \in \lambda_\Omega(v)$. The label of t is a. Similarly as for places, we will denote by $\mathsf{trees}(u)$ the multiset of transitions Q upon which a newly constructed transition $u = (v, \alpha) \lhd Q \in \mathsf{T}^v_{\mathsf{new}}$ is based.

For a place p and transition u in $\mathsf{T}^v_{\mathsf{new}}$, the weight $W_{\Omega(\Sigma)}(p, u)$ is given by:

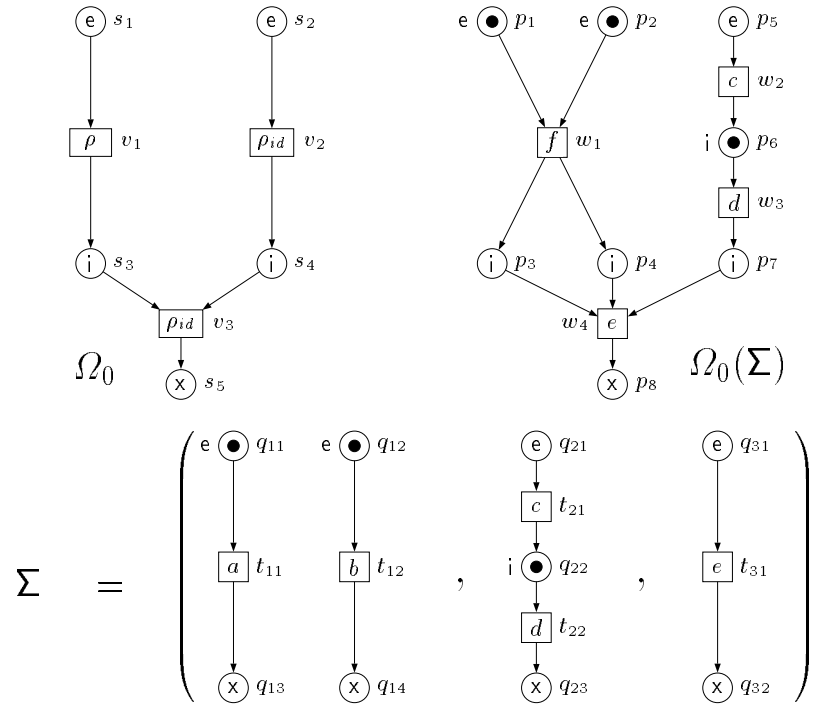
$$\sum\nolimits_{z \in \mathsf{trees}^v(p)} \sum\nolimits_{t \in \mathsf{trees}(u)} W_{\Sigma_v}(z, t) \cdot Q(t)$$

The weight $W_{\Omega(\Sigma)}(u, p)$ is defined similarly.

Theorem 1. *$\Omega(\Sigma)$ is a static or dynamic box.* □

Perhaps surprisingly, the above result is not straightforward [6].

A running Example. Figure 3 shows an operator box Ω which will serve as a running example. Ω_0 is a simple, pure, safe and clean box. It is also factorisable, which can be checked by inspecting all the complex markings reachable from the entry or exit marking, and all its transitions are reversible. For example, $({}^\circ\Omega_0, \emptyset) = (\{s_1, s_2\}, \emptyset)$ has a unique factorisation, $(\{v_1, v_2\}, \emptyset, \emptyset, \{v_3\})$, and the reachable complex marking $(\{s_1\}, \{v_2\})$ has a unique factorisation as well, $(\{v_1\}, \{v_2\}, \emptyset, \{v_3\})$. In all, fact_{Ω_0} comprises 13 factorisations. One can see that $\mathsf{dom}_{\Omega_0} = \mathsf{Box}^d \times \mathsf{Box}^d \times \mathsf{Box}^s \cup \mathsf{Box}^s \times \mathsf{Box}^s \times \mathsf{Box}^d \cup \mathsf{Box}^s \times \mathsf{Box}^s \times \mathsf{Box}^s$. Figure 3 shows an Ω_0-tuple of boxes, $\Sigma = (\Sigma_{v_1}, \Sigma_{v_2}, \Sigma_{v_3})$ whose factorisation, $(\{v_1\}, \{v_2\}, \emptyset, \{v_3\})$, belongs to fact_{Ω_0}, and the tuple itself belongs to the domain of application of the operator box Ω_0. The box $\Omega_0(\Sigma)$ exemplifies net refinement, and the full linear notation for its place and transition (tree) names is also shown in figure 3.



$p_1 = s_1 \lhd v_1 \lhd q_{11}$ $\quad p_2 = s_1 \lhd v_1 \lhd q_{12}$ $\quad w_1 = (v_1, f) \lhd \{t_{11}, t_{12}\}$

$p_3 = s_3 \lhd \{v_1 \lhd q_{13}, v_3 \lhd q_{31}\}$ $\quad p_4 = s_3 \lhd \{v_1 \lhd q_{14}, v_3 \lhd q_{31}\}$ $\quad w_2 = (v_2, c) \lhd t_{21}$

$p_5 = s_2 \lhd v_2 \lhd q_{21}$ $\quad p_6 = v_2 \lhd q_{22}$ $\quad w_3 = (v_2, d) \lhd t_{22}$

$p_7 = s_4 \lhd \{v_2 \lhd q_{23}, v_3 \lhd q_{31}\}$ $\quad p_8 = s_5 \lhd v_3 \lhd q_{32}$ $\quad w_4 = (v_3, e) \lhd t_{31}$

Fig. 3. Boxes of the running example $(\rho = \rho_{id} \backslash \{(\{a\}, a), (\{b\}, b)\} \cup \{(\{a, b\}, f)\})$.

Further restrictions on the Domain of an Operator Box. As we already mentioned, we have removed some restrictions imposed previously on plain and operator boxes, such as ex-directedness. The motivation for such a change came from the desire to accommodate within the general scheme operator boxes which can express iterative behaviours in an appealing way, such as Ω^* in figure 4 which models the behaviour $v_1^* v_2$ (in other words, any number of repetitions of the net refining v_1 followed by a behaviour of the net refining v_2). It is worth noting that Ω^* is useful in modelling guarded while-loops; basically, v_2 corresponds to the negation of the guard(s), and v_1 to the repetitive behaviour.

Allowing operator boxes like Ω^* does not create any problems as far as the consistency between nets and process expressions is concerned, but there is now a real danger of creating composite nets which are not in agreement with an intuitive meaning of operations specified by operator boxes. For consider the expression $(b^*c) \,\square\, a$. According to our intuition about the operation specified by Ω^*, one would expect that the net model of $(b^*c) \,\square\, a$ might look like the net in the middle of figure 4. But then it may be noticed that, from the entry marking, an evolution $\{b\}\{b\}\{a\}$ is allowed, which does not correspond to what one expects from a choice construct. The ex-directedness of boxes in the previous formulations of the box algebra was introduced in order to avoid this non-intuitive behaviour in the case of the choice operator. However, if it is ascertained that a loop does not occur initially in an enclosing choice, or in an enclosing loop, then the problem also disappears, and ex-directedness is no longer required. For example, a net model of $a;(b^*c)$, shown in the right of figure 4 is correct, which can be checked by considering all the behaviours from its entry marking. One could, of course, ask whether operators like Ω^* are important enough to justify the inevitable complications caused by their adoption within the general framework. The answer is yes, since they allow a crisp and faithful translation of looping constructs.

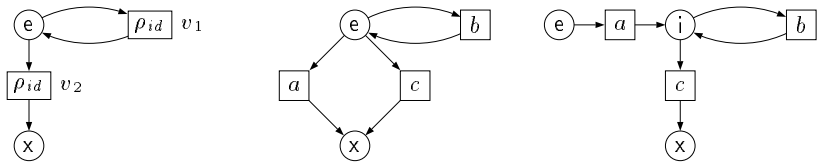


Fig. 4. Operator box Ω^* and nets modelling (tentatively) $(b^*c) \,\square\, a$ and $a;(b^*c)$.

As a consequence of the above discussion, we formulate two additional conditions on an Ω-tuple Σ in the domain of application of Ω.

Beh1 If Σ_v is non-e-directed then ${}^\bullet v \cap \Omega^\circ = \emptyset = {}^\bullet({}^\bullet v \cap {}^\circ\Omega)$, $({}^\bullet v)^\bullet = \{v\}$, and all the boxes Σ_w for $w \in {}^\bullet({}^\bullet v)$ are x-directed.

Beh2 If Σ_v is non-x-directed then $v^\bullet \cap {}^\circ\Omega = \emptyset = (v^\bullet \cap \Omega^\circ)^\bullet$, ${}^\bullet(v^\bullet) = \{v\}$, and all the boxes Σ_w for $w \in (v^\bullet)^\bullet$ are e-directed.

Adding (Beh1) and (Beh2) makes the problem described above disappear. For example, in $\Sigma \,\square\, \Theta$, both Σ and Θ should be ex-directed; and in $\Sigma;\Theta$, Σ

should be x-directed or Θ should be e-directed. Then one can see, for example, that a behaviour of $\Sigma \,\square\, \Theta$ is a behaviour of Σ or a behaviour of Θ, whereas a behaviour of $\Sigma; \Theta$ is a behaviour of Σ, or a terminated behaviour of Σ, followed by a behaviour of Θ.

In view of the conditions expressed by (Dom1–Dom2) and (Beh1–Beh2) it is relevant (and, as our experience shows, entirely satisfactory) to have the following sufficient conditions for e-directedness, x-directedness and ex-exclusiveness, in the refinement context.

Proposition 1. *If Ω and Σ_v, for every $v \in ({}^\circ\Omega)^\bullet$ (resp. $v \in {}^\bullet(\Omega^\circ)$), are all e-directed (resp. x-directed) boxes, then so is $\Omega(\Sigma)$.* □

Proposition 2. *Let Ω be an ex-exclusive operator box and suppose that, for every $v \in T_\Omega$ satisfying ${}^\bullet v \cap {}^\circ\Omega \neq \emptyset \neq v^\bullet \cap \Omega^\circ$ or $v^\bullet \cap {}^\circ\Omega \neq \emptyset \neq {}^\bullet v \cap \Omega^\circ$, it is the case that Σ_v is ex-exclusive. Then $\Omega(\Sigma)$ is ex-exclusive.* □

4 The Box Algebra

The *box algebra* is a *meta-model* parameterised by a set $\mathsf{ConstBox}$ of static and dynamic plain boxes which provide a denotational semantics of simple process expressions, and a disjoint set OpBox of operator boxes which provide interpretation for the connectives. The only assumption is that for all distinct $\Psi, \Theta \in \mathsf{ConstBox} \cup \mathsf{OpBox}$,

$$S_\Psi \cup T_\Psi \subseteq \mathsf{P_{root}} \cup \mathsf{T_{root}} \text{ and } S_\Psi \cap S_\Theta = T_\Psi \cap T_\Theta = \emptyset. \tag{2}$$

These assumptions are needed, for instance, to apply the results on solving recursive systems of equations on boxes obtained in [10].

Syntax We consider an algebra of process expressions over the signature $\mathsf{Const} \cup \{\overline{(.)}, \underline{(.)}\} \cup \{\mathsf{op}_\Omega \mid \Omega \in \mathsf{OpBox}\}$ where Const is a fixed non-empty set of *constants*, $\overline{(.)}$ and $\underline{(.)}$ are two unary operators, and each op_Ω is a connective (whose arity is the number of transitions of Ω). The set of constants is partitioned into the *static* constants, Const^s, and *dynamic* constants, Const^d; moreover, there are two disjoint subsets of Const^d, denoted by Const^e and Const^x, and respectively called the *entry* and *exit* constants. We will also use a fixed set Var of *process variables*.

We shall make use of four classes of process expressions corresponding to previously introduced classes of plain boxes: the *entry*, *dynamic*, *exit* and *static* expressions, denoted respectively by Expr^e, Expr^d, Expr^x and Expr^s. Collectively, we will refer to them as the *box expressions*, Expr^{alg}. We will also use a counterpart of the notion of the factorisation of a tuple of boxes. For an operator box Ω and an Ω-tuple of box expressions D (we again use the notation D_v to access individual members of D), we define the *factorisation* of D to be the tuple $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ such that $\mu_\delta = \{v \mid D_v \in \mathsf{Expr}^\delta\}$, for $\delta \in \{e, x, s\}$, and $\mu_d = \{v \mid D_v \in \mathsf{Expr}^d \setminus (\mathsf{Expr}^e \cup \mathsf{Expr}^x)\}$.

The syntax for the box expressions Expr^{alg} is given by:

$$\begin{array}{ll} \mathsf{Expr}^{s} & E ::= c^{s} \mid X \mid \mathsf{op}_{\Omega}(\mathsf{E}) \\ \mathsf{Expr}^{e} & F ::= c^{e} \mid \overline{E} \mid \mathsf{op}_{\Omega}(\mathsf{F}) \\ \mathsf{Expr}^{x} & G ::= c^{x} \mid \underline{E} \mid \mathsf{op}_{\Omega}(\mathsf{G}) \\ \mathsf{Expr}^{d} & H ::= c^{d} \mid F \mid G \mid \mathsf{op}_{\Omega}(\mathsf{H}) \end{array} \tag{3}$$

where $c^{\delta} \in \mathsf{Const}^{\delta}$, for $\delta \in \{e, x, s\}$, and $c^{d} \in \mathsf{Const}^{d} \setminus (\mathsf{Const}^{e} \cup \mathsf{Const}^{x})$ are constants; $X \in \mathsf{Var}$ is a process variable; $\Omega \in \mathsf{OpBox}$ is an operator box; and E, F, G and H are Ω–tuples of box expressions. The factorisations of E, F and G are respectively factorisations of the complex empty, entry and exit markings of Ω, and the factorisation of H is a factorisation of a marking reachable from the entry or exit marking of Ω different from ${}^{\circ}\Omega$ and Ω°. For every process variable $X \in \mathsf{Var}$, there is a unique defining equation $X \stackrel{\mathrm{df}}{=} \mathsf{op}_{\Omega}(\mathsf{L})$ where $\Omega \in \mathsf{OpBox}$ is an operator box and L is an Ω–tuple of process variables and static constants.

The above syntax does not incorporate the conditions (Dom1) and (Dom2). We could, of course, add these in a purely syntactic way, by taking advantage of the two results characterising compositionally defined ex-exclusive and x-directed boxes, i.e., propositions 1 and 2. However, we found it simpler to explicitly describe syntactic restrictions implied by (Dom1) and (Dom2), after giving a translation from the constant expressions to boxes.

Infinite Operators. If each operator box Ω in OpBox has finitely many transitions, the meaning of the syntax (3) is the usual one; it simply defines four sets of finite strings, or words. In the general case, however, we need to take into account Ω's with infinite transition sets, and one should ask what precisely do we then mean by the syntax (3) and the expressions it generates. Our answer is that expressions can, in general, be seen as trees and the syntax definition above as a definition of four sets of such trees. In what follows, we will use a still more general set of process expressions (we shall need this larger set of expressions later on, in the definition of a similarity relation on process expressions, and the definition of the inference rules of the operationa3l semantics) over the signature of the box algebra, denoted by Expr and referred to as *expressions*, defined by:

$$C \;::=\; c \mid X \mid \overline{C} \mid \underline{C} \mid \mathsf{op}_{\Omega}(\mathsf{C})$$

where $c \in \mathsf{Const}$ is a constant, $X \in \mathsf{Var}$ is a variable, and C is any Ω-tuple of expressions, for $\Omega \in \mathsf{OpBox}$. The precise meaning of expressions defined by this syntax is given by (possibly infinite) trees which do not have infinite paths from the root and are essentially syntax trees of the expressions they represent. The technical treatment is based on transfinite induction [6].

It follows directly from the syntax (3) and the assumptions we made that Expr^{e} and Expr^{x} are disjoint subsets of Expr^{d}, and that the static and dynamic expressions are disjoint sets; thus the factorisation of an Ω-tuple of box expressions is always a partition of the set of transitions of Ω, T_{Ω}.

It is convenient to have a notation for turning an expression in Expr into a corresponding static one. We again use $\lfloor H \rfloor$ to denote such an operation; it

removes from H all the occurrences of $\overline{(.)}$ and $\underline{(.)}$, and replaces every occurrence of each dynamic constant c by the corresponding static constant $\lfloor c \rfloor$ (i.e., we assume that each dynamic constant c is associated to a unique static constant, denoted $\lfloor c \rfloor$). The operators $\overline{(.)}$, $\underline{(.)}$ and $\lfloor . \rfloor$ can be applied in the usual way (i.e., elementwise) to sets as well as tuples of expressions. The same will be true of the semantical mapping box and the structural similarity relation $\equiv$, defined later on. In what follows, the expressions in $\mathsf{Const} \cup \overline{\mathsf{Const}^s} \cup \underline{\mathsf{Const}^s}$, i.e., those not involving any connective op_Ω nor a process variable, will be referred to as *flat*.

We will continue to use the boxes depicted in figure 3 in order to construct a simple yet illustrative algebra of process expressions. The *Do It Yourself (DIY) algebra* is based on two sets of boxes, $\mathsf{ConstBox} = \{\Phi_1, \Phi_{11}, \Phi_{12}\} \cup \{\Phi_2, \Phi_{21}, \Phi_{22}, \Phi_{23}\} \cup \{\Phi_3\}$ and $\mathsf{OpBox} = \{\Omega_0\}$, where $\lfloor \Phi_{ij} \rfloor = \Phi_i = \lfloor \Sigma_{v_i} \rfloor$ (for all i and j), $M_{\Phi_{11}} = \{q_{11}, q_{14}\}$, $M_{\Phi_{12}} = \{q_{13}, q_{12}\}$ and $M_{\Phi_{2k}} = \{q_{2k}\}$ (for $k = 1, 2, 3$). The constants of the DIY algebra correspond to the boxes in $\mathsf{ConstBox}$: $\mathsf{Const}^e = \{c_{21}\}$, $\mathsf{Const}^s = \{c_1, c_2, c_3\}$, $\mathsf{Const}^x = \{c_{23}\}$ and $\mathsf{Const}^d = \{c_{11}, c_{12}, c_{21}, c_{22}, c_{23}\}$. Moreover, $\lfloor c_{ij} \rfloor = c_i$, for every dynamic constant c_{ij}. The syntax of the DIY algebra is obtained by instantiating (3) with concrete constants and operator introduced above. For example, the syntax for the static and entry expressions is given respectively by $E ::= c_1 \mid c_2 \mid c_3 \mid X \mid \mathsf{op}_{\Omega_0}(E, E, E)$ and $F ::= c_{21} \mid \overline{E} \mid \mathsf{op}_{\Omega_0}(F, F, E)$.

4.1 Denotational Semantics of the Box Algebra

The denotational semantics is given in the form of a mapping box from box expressions to boxes. To begin with, constant expressions are mapped onto constant boxes of corresponding types, i.e., for every constant c and $\delta \in \{e, d, x, s\}$, $c \in \mathsf{Const}^\delta \;-\; \mathsf{box}(c) \in \mathsf{Box}^\delta \cap \mathsf{ConstBox}$. It is also assumed that, for every dynamic constant c, the underlying box is the same as for the corresponding static constant, i.e., $\lfloor \mathsf{box}(c) \rfloor = \mathsf{box}(\lfloor c \rfloor)$, and that for every (non-entry and non-exit) dynamic box Σ reachable from an initially marked constant box there is a corresponding dynamic constant c, i.e., $\mathsf{box}(c) = \Sigma$.

Syntactic restrictions resulting from (Dom1) and (Dom2). We now introduce into the box expression framework the constraints (Dom1) and (Dom2) imposed on the tuples of plain boxes in the domain of application of an operator box. We assume that within the set of box expressions generated by the syntax (3), three additional sets of expressions have been defined: the set of *well formed* expressions, Expr^{wf}, which comprises all flat expressions, among others; the set of *ex-exclusive* expressions, $\mathsf{Expr}^{excl} \subseteq \mathsf{Expr}^{wf}$; and the set of *x-directed* expressions, $\mathsf{Expr}^{xdir} \subseteq \mathsf{Expr}^{wf}$, in such a way that they are the largest sets of expressions such that the following are satisfied:

- For every constant c in Expr^{excl} or Expr^{xdir}, $\mathsf{box}(c)$ is ex-exclusive or x-directed, respectively.
- For all expressions $\overline{E}$ and $\underline{E}$ in Expr^{wf}, Expr^{excl} or Expr^{xdir}, the expression E belongs to Expr^{wf}, Expr^{excl} or Expr^{xdir}, respectively.

- For every variable X in Expr^{wf}, Expr^{excl} or Expr^{xdir} defined by $X \stackrel{\text{df}}{=} \mathsf{op}_\Omega(\mathsf{L})$, the expression $\mathsf{op}_\Omega(\mathsf{L})$ belongs to Expr^{wf}, Expr^{excl} or Expr^{xdir}, respectively.
- For every well formed expression $\mathsf{op}_\Omega(\mathsf{D})$, the expressions in D are also well formed and, for every $v \in T_\Omega$, if ${}^\bullet v \cap v^\bullet \neq \emptyset$ then D_v is ex-exclusive, and if v is not reversible then D_v is x-directed.
- For every ex-exclusive expression $\mathsf{op}_\Omega(\mathsf{D})$, Ω is an ex-exclusive operator box such that, for every $v \in T_\Omega$ satisfying ${}^\bullet v \cap {}^\circ\Omega \neq \emptyset \neq v^\bullet \cap \Omega^\circ$ or $v^\bullet \cap {}^\circ\Omega \neq \emptyset \neq {}^\bullet v \cap \Omega^\circ$, D_v is ex-exclusive.
- For every x-directed expression $\mathsf{op}_\Omega(\mathsf{D})$, Ω is an x-directed box and the expression D_v, for every $v \in {}^\bullet(\Omega^\circ)$, is x-directed.

Restrictions similar to those formulated above can also be introduced to render the conditions (Beh1) and (Beh2).

If all operator boxes in OpBox are pure and have only reversible transitions, then we can take Expr^{wf} to be the entire set of box expressions defined by the syntax (3), i.e., $\mathsf{Expr}^{wf} = \mathsf{Expr}^{alg}$. It is both interesting and relevant to observe that the standard PBC [4] can be treated in this way; moreover, this is also the case for the DIY algebra.

To simplify the presentation, in what follows we will implicitly assume that Expr^{alg} contain only well formed expressions.

Variables. With each defining equation $X \stackrel{\text{df}}{=} \mathsf{op}_\Omega(\mathsf{L})$, we associate an equation on boxes $X \stackrel{\text{df}}{=} \Omega(\Delta)$ where $\Delta_v = L_v$ if L_v is a process variable (treated here as a net variable), and $\Delta_v = \mathsf{box}(L_v)$ if L_v is a static constant. This creates a system of *equations on boxes* of the following form (one equation for every variable X in Var):

$$X \stackrel{\text{df}}{=} \Omega_X(\Delta_X). \tag{4}$$

On the right-hand side, Ω_X is an operator box with the empty marking, and Δ_X is an Ω_X-tuple whose elements are either recursion variables or static boxes. Now, given a mapping $\mathsf{sol} : \mathsf{Var} \to \mathsf{Box}^s$ assigning a static box to every variable in Var, we will denote by $\Delta_X[\mathsf{sol}]$ the Ω_X-tuple obtained from Δ_X by replacing each variable Y by $\mathsf{sol}(Y)$. Then, a *solution* of the system of equations (4) is an assignment sol such that, for every variable $X \in \mathsf{Var}$, $\mathsf{sol}(X) = \Omega_X(\Delta_X[\mathsf{sol}])$, where '=' denotes *equality* on nets. It turns out [3, 6, 10, 16] that the system of net equations (4) always has at least one solution and that all the solutions have the same behaviour since the corresponding static boxes have isomorphic transition systems (the latter will follow from theorem 6 below). We then fix *any* such solution sol and define, for every process variable X, $\mathsf{box}(X) = \mathsf{sol}(X)$.

Compound Expressions. The definition of box is completed by considering all the remaining static and dynamic expressions. For every box expression $\mathsf{op}_\Omega(\mathsf{D})$ and every static expression E, $\mathsf{box}(\mathsf{op}_\Omega(\mathsf{D})) = \Omega(\mathsf{box}(\mathsf{D}))$, $\mathsf{box}(\overline{E}) = \overline{\mathsf{box}(E)}$ and $\mathsf{box}(\underline{E}) = \underline{\mathsf{box}(E)}$.

Theorem 2. *For every well-formed box expression D, $\mathsf{box}(D)$ is a static or dynamic box such that, for every $\delta \in \{e, d, x, s\}$, $D \in \mathsf{Expr}^\delta \;-\; \mathsf{box}(D) \in \mathsf{Box}^\delta$. Moreover, if D is an ex-exclusive or x-directed expression, then $\mathsf{box}(D)$ is an ex-exclusive or x-directed plain box, respectively.* □

In the case of the DIY algebra, we define the box mapping by setting, for every static constant c_i, $\mathsf{box}(c_i) = \Phi_i$, and for every dynamic constant c_{ij}, $\mathsf{box}(c_{ij}) = \Phi_{ij}$. Other than that, we follow the general definitions. E.g., the box in figure 3 can be derived thus: $\mathsf{box}(\mathsf{op}_{\Omega_0}(\overline{c_1}, c_{22}, c_3)) = \Omega_0(\mathsf{box}(\overline{c_1}), \mathsf{box}(c_{22}), \mathsf{box}(c_3)) = \Omega_0(\overline{\mathsf{box}(c_1)}, \Phi_{22}, \Phi_3) = \Omega_0(\overline{\Phi_1}, \Phi_{22}, \Phi_3) = \Omega_0(\Sigma_{v_1}, \Sigma_{v_2}, \Sigma_{v_3}) = \Omega_0(\Sigma)$.

4.2 Structural Similarity relation on Expressions

A structural similarity relation on box expressions, $\equiv$, provides a partial structural identification of the box expressions with the same denotational semantics. We introduced a larger set of expressions than it was strictly necessary to ensure that the rules of the structural equivalence (and, later, operational semantics) act as term rewriting rules. That is, whenever a box expression can match one side of such a rule, then it should be guaranteed that the other side is a box expression of the correct type too. Hence, we need to allow provisionally more general expressions, such as $\overline{\overline{X}}$, $\underline{c}$, $\overline{D}$ and $\mathsf{op}_\Omega(\mathsf{C})$, where C does not correspond to a factorisation in fact_Ω, just to be able to express and prove properties saying that we do not need them actually. Formally, we define $\equiv$ to be the least binary relation on expressions in Expr such that (5)—(9) below hold.

- For all expressions D, H and J,

$$\boxed{D \equiv D \qquad \frac{D \equiv H}{H \equiv D} \qquad \frac{D \equiv J \equiv H}{D \equiv H}} \tag{5}$$

- For all flat expressions D and H satisfying $\mathsf{box}(D) = \mathsf{box}(H)$, and all equations $X \stackrel{\mathrm{df}}{=} \mathsf{op}_\Omega(\mathsf{L})$,

$$\boxed{D \equiv H \qquad X \equiv \mathsf{op}_\Omega(\mathsf{L}).} \tag{6}$$

- For every operator box Ω in OpBox and all factorisations μ and κ of, respectively, ${}^\circ\Omega$ and Ω°:

$$\boxed{\mathsf{op}_\Omega(\mathsf{D}) \equiv \overline{\mathsf{op}_\Omega(\mathsf{H})} \qquad \mathsf{op}_\Omega(\mathsf{J}) \equiv \underline{\mathsf{op}_\Omega(\mathsf{H})}} \tag{7}$$

  where D, J and H are Ω-tuples of expressions such that, for every $v \in T_\Omega$, $D_v = \overline{H_v}$ if $v \in \mu_e$ and $D_v = H_v$ otherwise; and $J_v = \underline{H_v}$ if $v \in \kappa_x$ and $J_v = H_v$ otherwise.
- For every operator box Ω in OpBox, for every complex marking reachable from the entry marking of Ω, different from ${}^\circ\Omega$ and Ω°, and for every pair of different factorisations μ and κ of that marking,

$$\boxed{\mathsf{op}_\Omega(\mathsf{D}) \equiv \mathsf{op}_\Omega(\mathsf{H})} \tag{8}$$

  where D and H are Ω-tuples of expressions for which there is an Ω-tuple of expressions[2] C such that, for every $v \in T_\Omega$, $D_v = \overline{C_v}$ if $v \in \mu_e$, $D_v = \underline{C_v}$ if

[2] The tuple C used in the formulation of (8) intuitively corresponds to the 'common' part of D and H.

$v \in \mu_x$ and $D_v = C_v$ otherwise; and $H_v = \overline{C_v}$ if $v \in \kappa_e$, $H_v = \underline{C_v}$ if $v \in \kappa_x$ and $H_v = C_v$ otherwise.

– For all expressions D and H, for every operator box Ω in OpBox, and for all Ω-tuples of expressions D and H:

$$\boxed{\frac{D \equiv H}{\overline{D} \equiv \overline{H}} \qquad \frac{D \equiv H}{\underline{D} \equiv \underline{H}} \qquad \frac{\mathsf{D} \equiv \mathsf{H}}{\mathsf{op}_\Omega(\mathsf{D}) \equiv \mathsf{op}_\Omega(\mathsf{H})}} \tag{9}$$

The meaning of the similarity relation $\equiv$ is standard if we only consider operator boxes OpBox with finite transition sets. In the general case, we use a technique similar to that employed in the case of syntactic definition of expressions.

The structural similarity relation is closed in the domain of box expressions[3] and preserves the types of expressions it relates.

Theorem 3. *If D and H are box expressions, then $D \equiv H$ if and only if $\lfloor D \rfloor \equiv \lfloor H \rfloor$ and $\mathsf{box}(D) = \mathsf{box}(H)$.* □

Thus $\equiv$ is a sound equivalence notion from the point of view of denotational semantics; it is also complete in the sense that $\mathsf{box}(D) = \mathsf{box}(H)$ implies $D \equiv H$ provided that $\lfloor D \rfloor \equiv \lfloor H \rfloor$. The latter condition cannot be left out and, in terms of the DIY algebra, a counterexample is provided by the expressions $D = \overline{X}$ and $H = \overline{Y}$, where X and Y are variables defined by $X \stackrel{\text{df}}{=} \Omega_0(c_1, c_1, X)$ and $Y \stackrel{\text{df}}{=} \Omega_0(c_1, c_1, Y)$.

The DIY algebra gives rise to five specific rules for the structural equivalence relation (we omit here their symmetric, hence redundant, counterparts). The first two are derived from (6): $\overline{c_2} \equiv c_{21}$ and $\underline{c_2} \equiv c_{23}$. The third and fourth are derived from (7): $\mathsf{op}_{\Omega_0}(\overline{E}, \overline{F}, G) \equiv \overline{\mathsf{op}_{\Omega_0}(E, F, G)}$ and $\mathsf{op}_{\Omega_0}(E, F, \underline{G}) \equiv \underline{\mathsf{op}_{\Omega_0}(E, F, G)}$. There is a single instance of (8): $\mathsf{op}_{\Omega_0}(\underline{E}, \underline{F}, G) \equiv \mathsf{op}_{\Omega_0}(E, F, \overline{G})$. But also $\mathsf{op}_{\Omega_0}(\overline{E}, \overline{F}, \overline{G}) \equiv \mathsf{op}_{\Omega_0}(\overline{E}, \overline{F}, G)$ which shows why it was necessary to use a larger set of expressions than Expr^{alg}. An application of these rules (more precisely, rules (5) and (9)) is illustrated by the following derivation tree:

$$\dfrac{\overbrace{\dfrac{\underbrace{\underline{c_1} \equiv \underline{c_1} \qquad c_{23} \equiv \underline{c_2} \qquad c_3 \equiv c_3}_{\mathsf{op}_{\Omega_0}(\underline{c_1}, c_{23}, c_3) \equiv \mathsf{op}_{\Omega_0}(\underline{c_1}, \underline{c_2}, c_3)} \qquad \mathsf{op}_{\Omega_0}(\underline{c_1}, \underline{c_2}, c_3) \equiv \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3})}^{\mathsf{op}_{\Omega_0}(\underline{c_1}, c_{23}, c_3) \equiv \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3})}}$$

4.3 Transition rules of the Operational Semantics

We now introduce operational rules based on the transitions of boxes which provide the denotational semantics of the box algebra expressions. Consider

[3] That is, whenever a box expression can match one side of a rule, then it is also guaranteed that the other side is a box expression too. Similar comment applies to the rules of the structured operational semantics.

the set of all the transition (tree) names in the boxes derived through the box mapping

$$\mathsf{T}^{alg}_{\mathsf{tree}} \;=\; \bigcup_{D \,\in\, \mathsf{Expr}^{alg}} T_{\mathsf{box}(D)} \;=\; \bigcup_{E \,\in\, \mathsf{Expr}^{s}} T_{\mathsf{box}(E)}\,.$$

One can see that each $t \in \mathsf{T}^{alg}_{\mathsf{tree}}$ has a unique label, $\mathsf{lab}(t)$, in all (possibly different) boxes associated with box expressions in which it occurs. The derivation system we shall define has moves of the form $D \stackrel{U}{\longrightarrow} H$, where D and H are expressions and U is a finite subset of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$. The idea here is that U is a valid step for the boxes associated with D and H, after augmenting them with the skip and redo transitions. We will denote, for every such set, $\mathsf{lab}(U) = \sum_{t \in U} \{\mathsf{lab}(t)\}$ assuming that $\mathsf{lab}(\mathsf{skip}) = \mathsf{skip}$ and $\mathsf{lab}(\mathsf{redo}) = \mathsf{redo}$. A move $D \stackrel{\emptyset}{\longrightarrow} H$ has a special interpretation: the empty step $U = \emptyset$ signifies that D and H are representations of the same system, and the only difference is that they present two possibly different views on it.

Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all the triples (D, U, H), where D and H are expressions and U is a finite subset of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$, such that (10)–(13) below hold. Note that instead of writing $(D, U, H) \in \longrightarrow$ we use the notation $D \stackrel{U}{\longrightarrow} H$.

– For every static expression E:

$$\boxed{\overline{E} \stackrel{\{\mathsf{skip}\}}{\longrightarrow} \underline{E} \qquad \underline{E} \stackrel{\{\mathsf{redo}\}}{\longrightarrow} \overline{E}.} \tag{10}$$

– For all box expressions D, J and H:

$$\boxed{\frac{D \equiv H}{D \stackrel{\emptyset}{\longrightarrow} H} \qquad \frac{D \stackrel{\emptyset}{\longrightarrow} J \stackrel{U}{\longrightarrow} H}{D \stackrel{U}{\longrightarrow} H} \qquad \frac{D \stackrel{U}{\longrightarrow} J \stackrel{\emptyset}{\longrightarrow} H}{D \stackrel{U}{\longrightarrow} H}} \tag{11}$$

– For every flat expression D and a non-empty step of transitions U enabled by $\mathsf{box}(D)$, there is a flat expression H such that $\mathsf{box}(D)\ [U\rangle\ \mathsf{box}(H)$ and:

$$\boxed{D \stackrel{U}{\longrightarrow} H.} \tag{12}$$

– For every $\Omega \in \mathsf{OpBox}$, and all Ω-tuples D and H of expressions:

$$\boxed{\frac{\forall\, v \in T \;:\; D_v \stackrel{U_v^1 \uplus \cdots \uplus U_v^{k_v}}{\longrightarrow} H_v}{\mathsf{op}_\Omega(\mathsf{D}) \stackrel{U}{\longrightarrow} \mathsf{op}_\Omega(\mathsf{H})} \quad \forall i \le k_v \;:\; (\mathsf{lab}(U_v^i), a_v^i) \in \lambda_\Omega(v)}$$

$$\text{where} \quad U \;=\; \textstyle\bigcup_{v \in T} \{(v, a_v^1) \lhd U_v^1, \ldots, (v, a_v^{k_v}) \lhd U_v^{k_v}\} \quad \text{is finite.} \tag{13}$$

Notice that the only way to generate a skip or redo is through applying the rules (10); i.e., if $D \xrightarrow{U} H$ then $\mathsf{skip} \in U$ implies $U = \{\mathsf{skip}\}$, and $\mathsf{redo} \in U$ implies $U = \{\mathsf{redo}\}$. The meaning of the operational semantics is standard if all the boxes in OpBox have finite transition sets; in general, however, we need to treat infinite operator boxes as well. The detailed treatment closely follows that for $\equiv$.

A move of the operational semantics transforms a box expression into another box expression with structurally equivalent underlying static expression, and the move generated is a valid step for the corresponding boxes. We interpret this as a result establishing the soundness of operational semantics.

Theorem 4. *Let D be a box expression and $D \xrightarrow{U} H$. Then H is a box expression such that $\mathsf{box}(D)_{\mathsf{sr}}\ [U\rangle\ \mathsf{box}(H)_{\mathsf{sr}}$ and $\lfloor D \rfloor \equiv \lfloor H \rfloor$.* □

The next result establishes the completeness of operational semantics.

Theorem 5. *Let D be a box expression and $\mathsf{box}(D)_{\mathsf{sr}}\ [U\rangle\ \Sigma_{\mathsf{sr}}$. Then there is a box expression H such that $\mathsf{box}(H) = \Sigma$ and $D \xrightarrow{U} H$.* □

In the DIY algebra, the axiomatisation of the flat expressions is given by the axioms: $\overline{c_1} \xrightarrow{\{t_{11}\}} c_{12}$, $\overline{c_2} \xrightarrow{\{t_{21}\}} c_{22}$, $\overline{c_1} \xrightarrow{\{t_{12}\}} c_{11}$, $c_{21} \xrightarrow{\{t_{21}\}} c_{22}$, $\overline{c_1} \xrightarrow{\{t_{11},t_{12}\}} \underline{c_1}$, $c_{22} \xrightarrow{\{t_{22}\}} \underline{c_2}$, $c_{12} \xrightarrow{\{t_{12}\}} \underline{c_1}$, $c_{22} \xrightarrow{\{t_{22}\}} c_{23}$, $c_{11} \xrightarrow{\{t_{11}\}} \underline{c_1}$ and $\overline{c_3} \xrightarrow{\{t_{31}\}} \underline{c_3}$, and the inference rule for the only operator box, Ω_0, can be formulated thus:

$$\frac{D \xrightarrow{\{t_1,u_1,\ldots,t_k,u_k,x_1,\ldots,x_l\}} D', H \xrightarrow{\{y_1,\ldots,y_m\}} H', G \xrightarrow{\{z_1,\ldots,z_n\}} G'}{\mathsf{op}_{\Omega_0}(D,G,H) \xrightarrow{U} \mathsf{op}_{\Omega_0}(D',G',H')}$$

where $k,l,m,n \geq 0$; $\{\mathsf{lab}(t_i),\mathsf{lab}(u_i)\} = \{a,b\}$, for every $i \leq k$; $\mathsf{lab}(x_i) \notin \{a,b\}$, for every $i \leq l$; and the step U is given by:

$$\{(v_1,f)\triangleleft\{t_1,u_1\},\ldots,(v_1,f)\triangleleft\{t_k,u_k\}\} \cup \{(v_1,\mathsf{lab}(x_1))\triangleleft x_1,\ldots,(v_1,\mathsf{lab}(x_l))\triangleleft x_l\} \cup$$
$$\{(v_2,\mathsf{lab}(y_1))\triangleleft y_1,\ldots,(v_2,\mathsf{lab}(y_m))\triangleleft y_m\} \cup \{(v_3,\mathsf{lab}(z_1))\triangleleft z_1,\ldots,(v_3,\mathsf{lab}(z_n))\triangleleft z_n\}$$

For example, the following is a valid sequence of three moves

$$\overline{\mathsf{op}_{\Omega_0}(c_1,c_2,c_3)} \xrightarrow{\{w_1,w_2\}} \mathsf{op}_{\Omega_0}(\underline{c_1},c_{22},c_3) \xrightarrow{\{w_3\}} \mathsf{op}_{\Omega_0}(c_1,c_2,\overline{c_3}) \xrightarrow{\{w_4\}} \underline{\mathsf{op}_{\Omega_0}(c_1,c_2,c_3)}.$$

A derivation tree for the first move is shown below:

$$\frac{\overline{\mathsf{op}_{\Omega_0}(c_1,c_2,c_3)} \xrightarrow{\{w_1,w_2\}} \mathsf{op}_{\Omega_0}(\underline{c_1},c_{22},c_3)}{\dfrac{\overline{\mathsf{op}_{\Omega_0}(c_1,c_2,c_3)} \xrightarrow{\emptyset} \mathsf{op}_{\Omega_0}(\overline{c_1},\overline{c_2},c_3)}{\mathsf{op}_{\Omega_0}(\overline{c_1},\overline{c_2},c_3) \xrightarrow{\left\{\begin{smallmatrix}(v_1,f)\triangleleft\{t_{11},t_{12}\}\\(v_2,c)\triangleleft t_{21}\end{smallmatrix}\right\}} \mathsf{op}_{\Omega_0}(\underline{c_1},c_{22},c_3)}}$$

$$\overline{c_1} \xrightarrow{\{t_{11},t_{12}\}} \underline{c_1} \qquad \overline{c_2} \xrightarrow{\{t_{21}\}} c_{22} \qquad c_3 \xrightarrow{\emptyset} c_3$$

5 The main Result

The consistency between the denotational and operational transition based semantics of box expressions will be expressed by making a statement about the transition systems they generate. Below, for every box expression D, $[D\rangle$ is the smallest set of expressions such that $D \in [D\rangle$, and if $H \xrightarrow{U} G$ and $H \in [D\rangle$ then $G \in [D\rangle$; moreover, $[D]_{\equiv}$ denotes the equivalence class of $\equiv$ containing D.

With each dynamic expression D we shall associate the *transition system* $\mathsf{ts}_D = (\mathsf{V}, \mathsf{L}, \mathsf{A}, [D]_{\equiv})$ such that: L — the set of all finite subsets of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$ — is the set of move labels; $\mathsf{V} = \{[H]_{\equiv} \mid H \in [D\rangle\}$ is the set of nodes; the arcs are given by $\mathsf{A} = \{([H]_{\equiv}, U, [G]_{\equiv}) \mid H \xrightarrow{U} G\}$; and $[D]_{\equiv}$ is the initial node. Moreover, with each static expression E, we shall associate the transition system $\mathsf{ts}_E = \mathsf{ts}_{\overline{E}}$.

We now can state a fundamental result which holds for the box algebra. In a nutshell, it proves that the operational and denotational semantics of a box expression capture the same behaviour, in arguably the strongest sense.

Theorem 6. *Let D be a box expression. Then ts_D and $\mathsf{ts}_{\mathsf{box}(D)}$ are isomorphic transition systems. Moreover,*

$$\mathsf{iso} = \{(\nu, \mathsf{box}(H)) \mid \nu \text{ is a node in } \mathsf{ts}_D \text{ and } H \in \nu\}$$

is an isomorphism for ts_D and $\mathsf{ts}_{\mathsf{box}(D)}$. □

The operational semantics based on transition trees is very expressive. However, it may sometimes be sufficient to record only the labels of the executed transitions, in the usual style of process algebras. Such a treatment can easily be accommodated within the scheme developed so far and a counterpart of theorem 6 formulated and proved correct [6]. The two types of operational semantics are related; in essence, each label based move is a transition based move with only transitions labels being recorded.

Theorem 6 extends also to partial order semantics based on Mazurkiewicz traces [17] – a model of partial order behaviour based solely on transitions. Briefly, given a step sequence ω and an independence relation ind, one can define a meaningful partial order $\mathsf{poset}_{\mathsf{ind}}(\omega)$ which is fully consistent with the standard causality semantics. For a labelled net Σ, $\mathsf{ind} = \mathsf{ind}_{\Sigma}$ is defined in the usual way; for the box expression model, it is possible to define a relation $\mathsf{ind}^{alg} \subseteq \mathsf{T}^{alg}_{\mathsf{tree}} \times \mathsf{T}^{alg}_{\mathsf{tree}}$ such that, for every box expression D, $\mathsf{ind}_{\mathsf{box}(D)} = (T_{\mathsf{box}(D)} \times T_{\mathsf{box}(D)}) \cap \mathsf{ind}^{alg}$. With such an ind^{alg} – a *global* independence relation – the consistency result obtained for transition based operational semantics can be lifted to the level of partial order executions, using the above property and theorem 6.

Acknowledgement

This research has been supported by the British-German Foundation ARC Project 1032 BAT (Box Algebra with Time).

References

1. J.Baeten, W.P.Weijland: *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18 (1990).
2. E.Best, R.Devillers: Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55:87-136 (1988).
3. E.Best, R.Devillers, J.Esparza: General Refinement and Recursion Operators for the Petri Box Calculus. Springer-Verlag, LNCS 665:130-140 (1993).
4. E.Best, R.Devillers, J.G.Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. Springer-Verlag, LNCS 609:21-69 (1992).
5. E.Best, R.Devillers, M.Koutny: Petri Nets, Process Algebras and Concurrent Programming Languages. Lectures on Petri Nets II: Applications, Advances in Petri Nets. Springer-Verlag, LNCS 1492:1-84 (1998).
6. E.Best, R.Devillers, M.Koutny: *Petri Net Algebra*. Book Manuscript (1999).
7. G.Boudol, I.Castellani: Flow Models of Distributed Computations: Event Structures and Nets. Rapport de Recherche, INRIA, Sophia Antipolis (1991).
8. P.Degano, R.De Nicola, U.Montanari: A Distributed Operational Semantics for CCS Based on C/E Systems. *Acta Informatica* 26:59-91 (1988).
9. R.Devillers: S-invariant Analysis of Recursive Petri Boxes. *Acta Informatica* 32:313-345 (1995).
10. R.Devillers, M.Koutny: Recursive Nets in the Box Algebra. Proc. of CSD'98 Conference, Fukushima, Japan, IEEE CS, 239-249 (1998).
11. U.Goltz: On Representing CCS Programs by Finite Petri Nets. Springer-Verlag, LNCS 324:339-350 (1988).
12. U.Goltz, R.Loogen: A Non-interleaving Semantic Model for Nondeterministic Concurrent Processes. *Fundamenta Informaticae* 14:39-73 (1991).
13. M.Hesketh, M.Koutny: An Axiomatisation of Duplication Equivalence in the Petri Box Calculus. Springer-Verlag, LNCS 1420:165-184 (1998).
14. C.A.R.Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
15. R.Janicki, P.E.Lauer: *Specification and Analysis of Concurrent Systems – the COSY Approach*. Springer-Verlag, EATCS Monographs on Theoretical Computer Science (1992).
16. M.Koutny, E.Best: Fundamental Study: Operational and Denotational Semantics for the Box Algebra. *Theoretical Computer Science* 211:1-83 (1999).
17. A.Mazurkiewicz: Trace Theory. Springer-Verlag, LNCS 255:279-324 (1987).
18. R.Milner: *Communication and Concurrency*. Prentice Hall (1989).
19. T.Murata: Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* 77:541-580 (1989).
20. E.-R.Olderog: *Nets, Terms and Formulas*. Cambridge Tracts in Th. Comp. Sci. 23 (1991).
21. G.Plotkin: A Structural Approach to Operational Semantics. DAIMI Technical Report FN-19, Computer Science Department, University of Aarhus (1981).
22. W.Reisig: *Petri Nets. An Introduction*. Springer-Verlag, EATCS Monographs on Theoretical Computer Science (1985).
23. D.Taubner: *Finite Representation of CCS and TCSP Programs by Automata and Petri Nets*. Springer-Verlag, LNCS 369 (1989).

Detection of Illegal Behaviors Based on Unfoldings

Jean-Michel Couvreur [1] and Denis Poitrenaud [2]

[1]*LaBRI, Université de Bordeaux I, Talence, France*
[2]*LIP6, Université de Paris 6, Paris, France*
Email: couvreur@labri.u-bordeaux.fr, Denis.Poitrenaud@lip6.fr

Abstract: We show how the branching process approach can be used for the detection of illegal behaviors. Our study is based on the specification of properties in terms of testers that cover safety as well as liveness properties. We demonstrate that the unfolding method can be used in this context and propose an extension of it, called unfolding graphs, for the computation of failure equivalent graphs.

Keywords: Model Checking, Partial Order, Failure Equivalence, Petri Net

1. Introduction

One of the main problems of exhaustive system verifications is the so called state explosion. Some methods ([Pel93], [God96], [Val91], [Val93]) avoid the exploration of the complete state space during verification operating reductions by using partial order semantics. These reductions are defined to preserve information necessary for verification. Another approach consists in implementing the verification directly on a representation of the partial order. Branching processes (or unfoldings) ([NPW81], [Eng91]) propose branching semantics of partial orders for Petri nets. McMillan [McM92] has described the construction of a finite branching process in which all reachable markings are represented. He has shown how branching processes can be used for the detection of global deadlocks and the reachability problem for finite state space Petri nets. Many improvements of this method concern the efficiency of the unfolding construction and the verification of safety properties, or the verification of branching temporal logic [Esp93] and linear temporal logic [CP96], [Poi96], [Wal98].

We present how the branching process approach can be used for the detection of illegal behaviors. Our study is based on the specification of properties in terms of testers [Val93] which cover safety and liveness properties. We demonstrate that the unfolding method can be used in this context and propose an extension of it, called unfolding graphs, for the computation of failure equivalent graphs [Val90].

After a presentation of the Petri net model and the associated partial order semantics, we recall the notion of tester and the corresponding verification process. In the sequel, we discuss the use of unfolding in this context and present a simple characterization of illegal behaviors. Then, unfolding graphs are introduced and discussed. Finally, a section is dedicated to the experimentation of the different methods and some concluding remarks close the paper.

2. Preliminaries

The first part of this section presents some basic notions and notations of P/T net, processes, and branching processes. The second part describes how properties of a system can be expressed by means of testers.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp.364-383, 1999.
C Springer-Verlag Berlin Heidelberg 1999

P/T Nets and their behaviors

We assume that the reader is familiar with the basic notions and notations of P/T nets as given for instance in [Mur89]. We denote a P/T net by $N = (P, T, F, m_0)$ where (P, T, F) is a net and m_0 is its initial marking. The set of predecessors of a transition t is called its *precondition* and is denoted by ${}^\bullet t$. Its set of successors is called its *postcondition* and is denoted by $t^\bullet$. For a given marking, a transition is said to be *enabled* if each place of its precondition contains at least one token. The firing of a transition produces a new marking by removing one token from each place of its precondition and by adding one in each place of its postcondition. The set of reachable markings of a net N is denoted by $Reach(N)$. The transition system of a P/T net N is $(Reach(N), T, F, m_0)$ where $F = \{(m, t, m') \mid m(t>m'\}$. By extension, a labeled P/T net N is denoted by $(P, T, \Sigma, \lambda, F, m_0)$ where Σ is the action set and λ is an application from T to Σ. It also defines a transition system.

An alternative to this representation is to describe system behaviors by particular labeled P/T nets called *processes*. In a process, places and transitions are respectively called *conditions* and *events*. A process is a labeled P/T net for which each condition has at most one input and one output event. Conditions without input event are said to be *initial* and those without output events are said to be *final*. The initial process of a net is formed by a set of conditions, one for each token of the initial marking. The label of a condition corresponding to a token is the place of the original net containing it. It is clear that in the initial process, each condition is initial and final. A new process is constructed from a given process by connecting to some final conditions of the process a new event labeled t with its output conditions. To be valid, the labels of the input and output conditions must coincide with the input and output places of the transition t of the original net. It is important to note that a process is cycle free by definition and that the notion of process takes into account the concurrence of the transition firings.

The construction of all the possible behaviors of a net leads to the construction of a new type of labeled P/T net called a *branching process*. In a branching process, each condition has at most one input event and the net is cycle free. Conditions without input events are called *initial*. Processes of a branching process are their sub graphs for which at each condition has at most one output event and such that the set of conditions without input events is exactly the set of initial conditions of the branching process. A given branching process can be extended by the extension of one of its processes. This operation is valid only if there does not exist a distinct event with the same input conditions and the same label in the branching process. This construction can lead to an infinite branching process even in the case of a finite state system. The construction of a particular finite prefix of the complete branching process is the starting point of the unfolding techniques presented in following sections.

We denote a (branching) process of a net $N = (P, T, F, m_0)$ by $U = (B, E, W, h)$ where B and E are the sets of conditions and events respectively, W is the flow relation and h is a labeling function from $B \cup E$ to $P \cup T$. To be a valid (branching) process of N, U must satisfy the constraints above.

As previously seen, processes and branching processes are cycle-free graphs by definition. This characteristic induces a partial order relation $\leq$ on the nodes (conditions and events) of the graph. In this order, the initial conditions are the smallest elements (denoted by $Min(U)$). Each pair (x, y) of nodes of a branching process belongs to precisely one of the following relations:

- *Causality relation*: x is before y ($x \leq y$) or x is after y ($x \geq y$),
- *Conflict relation* ($x \# y$): two distinct events, one before x and the other before y, share an input condition,
- *Concurrency relation* ($x // y$): x and y are neither in conflict nor causal.

The causality relation $x \leq y$ between two events reveals that in any process where y can be fired, the event x is fired first. Two events in conflict can not be fired in the same process, while two concurrent events can be fired in parallel. The relations on the conditions are interpreted in a similar way. In particular, a set of concurrent conditions can be marked in the same state. In a process, two nodes are either concurrent or causal.

A *configuration* is a set of conflict-free events (no pair of events belongs to the conflict relation) and closed by the causality relation (each event which is before any event of the configuration belongs to the configuration). Configurations exactly describe the processes of an unfolding: the set of conditions and events constructed from the events of a configuration completed by their surrounding conditions and the initial ones is a process; respectively events of a process form a configuration. We denote by *Conf(U)*, the set of configurations of a (branching) process U and by $MaxConf(U) \subseteq Conf(U)$ the set of maximal configurations (w.r.t. set inclusion). The minimal configuration (w.r.t. set inclusion) which contains an event e is called the *local configuration* of e and is denoted $[e]$. By extension, for a condition b, we denote by $[b]$ the local configuration $[{}^\bullet b]$ if b has an input event, and the empty set otherwise. Moreover, for a set $A \subseteq B \cup E$, $[A]$ denotes the event set obtained as the union of local configurations of the elements of A. Note that $[A]$ is not necessarily a configuration. For a configuration C, we denote by $Cut(C)$ the condition set defined by $(Min(U) \cup C^\bullet) \setminus {}^\bullet C$. It is clear that $h(Cut(C))$ corresponds to the marking of N reached after the firing of $h(C)$.

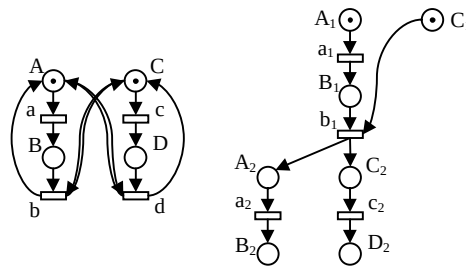


Figure 1: *A P/T net and one of its process*

The Figure 1 presents a P/T net and one of its processes. One can note that this process is maximal and leads to a dead marking. The Figure 2 presents the (infinite) complete branching process of the net and a finite branching process called the *unfolding*. This unfolding has some "good properties." In particular, it is possible to demonstrate the reachability of a marking from the initial one and the presence of dead markings.

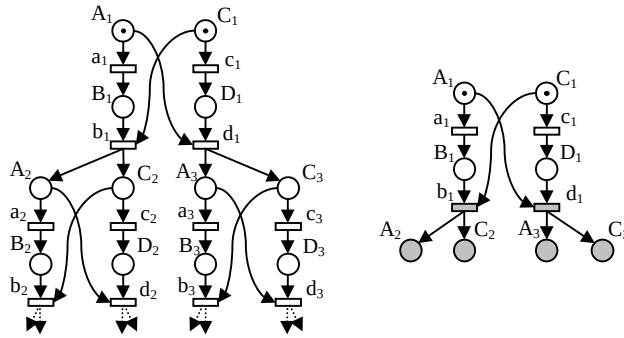


Figure 2: *The complete branching process and one unfolding of the net.*

In the sequel we use the following particular notations:

- $\beta = (B(\beta), E(\beta), W(\beta), h(\beta))$ is used to denote the different components of branching process β.
- $\Pi(C)$ is used to denote the process corresponding to a configuration C and $\Pi(C\backslash C1)$ when $C1 \subseteq C$ denotes the process from $Cut(C1)$ composed by the events $C\backslash C1$.
- $\pi 1 \preceq \pi 2$ means that process $\pi 1$ is a prefix of process $\pi 2$.
- $\pi 1.\pi 2$ is a concatenation of two processes such that $h(Cut(\pi 1)) = h(Min(\pi 2))$.
- e_0 is the artificial initial event which produces the initial marking of any branching process.

Tester and failure equivalence

A tester [Val93] is an automaton that controls the behavior of a labeled P/T net through a subset of actions said to be controlled. The edges of the tester are labeled by controlled actions. Hence, a P/T net behaves without constraint for transitions labeled by uncontrolled actions and is synchronized with the tester for transitions labeled with controlled actions. Büchi automata are a particular kind of testers for which all the actions are controlled.

The tester states are typed (mortal reject, deadlock reject, live reject and divergent reject) and lead to the detection of illegal behaviors:

1. *Mortal illegal behavior*: behavior leading the tester into a state of *mortal reject*.
2. *Deadlock illegal behavior*: deadlock behavior (which can not be extended) leading the tester into a state of *deadlock reject*.
3. *Live illegal behavior*: Infinite behavior which leads the tester an infinite number of times into a state of *live reject*.
4. *Divergent illegal behavior*: Infinite behavior which leads (and leaves) the tester into a state of *divergent reject.*

Detection of illegal behaviors of types 1 and 2 involve reachability problems and behaviors of types 3 and 4 involve cycle detection in the reachability graph.

Definition *(tester)*: A tester Z is a tuple $(Q, Control, H, q_0, Mortal, Deadlock, Live, Divergent)$ where

- Q is a set of states,
- *Control* is a set of controlled actions,

- H is a transition relation ($H \subseteq Q \times Control \times Q$),
- q_0 is an initial state ($q_0 \in Q$),
- *Mortal, Deadlock, Live* and *Divergent* are subsets of Q.

The Figure 3 gives a symbolic notation for qualifing the type of the tester states and presents some simple tester samples. Tester 1 detects the deadlock behaviors of a net. Tester 2 exhibits behaviors which are not in the language $(ab)^*+(ab)^*a$. Tester 3 detects divergent behaviors of the form *ababab … ba.*

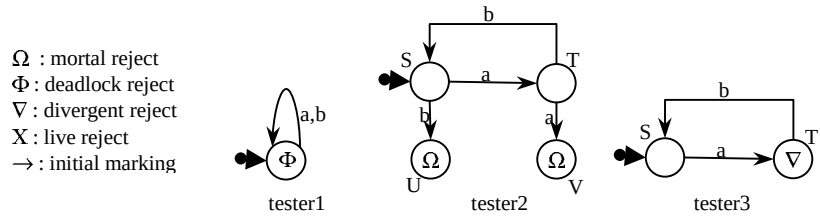


Figure 3: *Tester samples*

The synchronization of a tester with a transition system gives a new transition system defined as follows.

Definition *(synchronization)*: Let $Z = (Q, Control, H, q_0, Mortal, Deadlock, Live, Divergent)$ be a tester. Let $Sys = (S, \Sigma, F, s_0)$ be a transition system such that $Control \subseteq \Sigma$. Define Z // Sys to be the transition system (S', Σ', F', s_0') such that

- $S' = Q \times S$
- $\Sigma' = \Sigma$
- $F' = \{((q_1, s_1), a, (q_2, s_2)) \in S' \times \Sigma' \times S' : (q_1, a, q_2) \in H \wedge (s_1, a, s_2) \in F\} \cup$
  $\{((q, s_1), a, (q, s_2)) \in S' \times \Sigma' \times S' : a \notin Control \wedge (s_1, a, s_2) \in F\}$
- $s_0' = (q_0, s_0)$

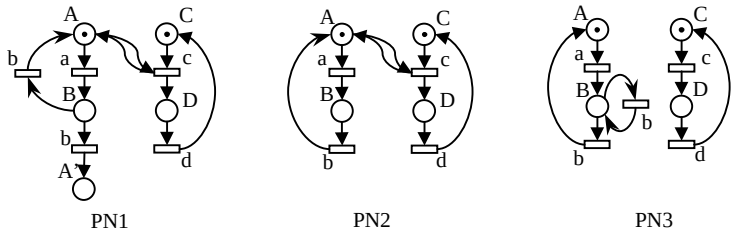


Figure 4: *Samples of labeled P/T nets*

Illegal behavior can be defined for a synchronization.

Definition *(illegal behavior)*: Let Z be a tester and Sys be a transition system. Let $\sigma = (q_0, s_0)\ a_0\ (q_1, s_1)\ a_1 \ldots$ be a finite ($i \in [0, n]$) or infinite ($i \in \mathbb{N}$) behavior of Z // Sys. We say

1. σ is a mortal reject behavior iff $\exists i: q_i \in Mortal$,
2. σ is a deadlock reject behavior iff σ is finite $\wedge\ q_n \in Deadlock \wedge (q_n, s_n)^\bullet = \emptyset$,
3. σ is a live reject behavior iff $\{i \mid q_i \in Live\}$ and $\{i \mid a_i \in Control\}$ are infinite, and

4. σ is a divergent reject behavior iff σ is infinite $\wedge \exists i_0$: $q_{i0} \in$ *Divergent* $\wedge \forall i > i_0$, $a_i \notin$ *Control.*

By synchronizing testers of Figure 3 with labeled P/T nets of Figure 4, we obtain the table of Figure 5. A "reject" specifies that tester has detected an illegal behavior.

	Tester1	Tester2	Tester3
PN1	Reject		
PN2			
PN3		reject	reject

Figure 5: *Tester results*

The synchronization of a tester *Z* can also be directly done on a labeled P/T net *N*. The synchronization consists in the construction of a new P/T net, denoted *Z // N*. It corresponds to the fusion of transitions with regard to controlled actions followed by the association of a place of the new P/T net *Z // N* to each state of the tester. We denote by *Mortal(Z // N)*, *Deadlock(Z // N)*, *Live(Z // N)* and *Divergent(Z // N)* the corresponding sets of reject places and *Tester(Z // N)* the tester place set.

The four forms of illegal behaviors are translated in the following way:
1. *Mortal illegal behavior*: behavior which marks a place of *mortal reject*,
2. *Deadlock illegal behavior*: finite behavior which the final marking is a deadlock and contains a marked place of *deadlock reject*,
3. *Live illegal behavior*: Infinite behavior modifying infinitely often the marking of a place of *live reject*, and
4. *Divergent illegal behavior*: Infinite behavior which marks and leaves marked a place of *divergent reject.*

Moreover, illegal behavior can be expressed in terms of processes.

Definition *(illegal process)*: Let *Z* be a tester and *N* be a labeled P/T net. Let π be a finite or infinite process of *Z // N*. Then
1. π is a mortal reject process iff $\exists b \in B(\pi)$: $h(b) \in$ *Mortal(Z // N)*,
2. π is a deadlock reject process iff π is finite and maximal $\wedge$ $h(Cut(\pi)) \cap$ *Deadlock(Z // N)* $\neq \emptyset$,
3. π is a live reject process iff $\{ b \in B(\pi) \mid h(b) \in$ *Live(Z // N)*$\}$ is infinite, and
4. π is a divergent reject process iff π is infinite $\wedge \exists b \in B(\pi)$: $h(b) \in$ *Divergent(Z // N)* $\wedge b^\bullet = \emptyset$.

The companion piece to the notion of tester is the failure equivalence [Val90]. Two P/T nets are *failure equivalent* with respect to a set of controlled actions iff the set of testers detecting an illegal behavior are the same for both nets. This equivalence relation is a congruence with respect to the synchronization operation on labeled P/T nets with respect to the set of controlled actions. We can remark that the nets of Figure 4 are not equivalent with respect to the set of controlled actions *{a, b}*. On the other hand, the nets *PN2* and *PN3* are equivalent with respect to the set *{c, d}*.

We introduce the notion of failure transition system: an abbreviation of transition system with respect to the failure equivalence that integrates the existence of deadlock and divergence behaviors local to a state.

Definition (*failure transition system*): A failure transition system *fsys* is a *tuple* $(S, \Sigma, F, s_0, Control, div, fail)$ where

- (S, Σ, F, s_0) is a transition system,
- *Control* is the controlled action set ($Control \subseteq \Sigma$),
- *div* is the divergent state set ($div \subseteq S$), and
- *fail* is an application from S to $2^{Control}$.

$s \in div$ means that the system can loop internally in the state s and $D \in fail(s)$ indicates that the system can reach internally a state in s where only transitions labeled by an element of D are enabled. We have to define an illegal behavior in terms of a failure transition system.

Definition *(illegal behavior)*: Let Z be a tester and *fsys* be a failure transition system. Let $\sigma = (q_0, s_0)\, a_0\, (q_1, s_1)\, a_1 \ldots$ be a finite ($i \in [0, n]$) or infinite ($i \in \mathbb{N}$) behavior of Z // *fsys*.

1. σ is a mortal reject behavior iff $\exists i: q_i \in Mortal$,
2. σ is a deadlock reject behavior iff σ is finite $\wedge\, q_n \in Deadlock \wedge \forall D \in fail(q_n)$: $\{a \in Control \cap D \mid (q_n, a, q) \in H\} = \emptyset$,
3. σ is a live reject behavior iff $\{i \mid q_i \in Live\}$ is infinite, and
4. σ is a divergent reject behavior iff (σ is infinite $\wedge \exists i_0: q_{i0} \in Divergent \wedge \forall i > i_0, a_i \notin Control$) $\vee$ ($\exists i: q_i \in Divergent \wedge s_i \in div$).

3. Unfolding

An unfolding of a P/T net is defined as a finite prefix of the complete branching process preserving a representation of all reachable markings. In this section, we present the usefulness of unfoldings for the detection of illegal behavior. Mortal and deadlock rejects are detected in a simple and natural way. Live and divergent rejects depend on the characterization of infinite behaviors. We expose the difficulties and the pitfalls of this characterization and present a simple technique for detecting these behaviors.

Elementary unfolding properties

The construction of an unfolding begins from the set of initial conditions and continues the enumeration of concurrent condition sets corresponding to the inputs of P/T net transitions. When such a new set is detected, a corresponding event is added as well as the associated output conditions.

At each state of the unfolding construction, we obtain a branching process β and a set of particular events *Cutout* for which there are extensions that have not been explored. The validity of this construction relies on the way that a process of a net can be projected in the unfolding.

1. *The process is contained in the unfolding*: the process is isomorphic to a configuration of the unfolding containing no cutout.
2. *The process crosses a cutout*: the local configuration of a cutout is isomorphic to a prefix of the process.

We call such a couple $(\beta, Cutout)$ a *stable branching process*.

Definition *(stable branching process)*: Let β be a branching process of a net N and *Cutout* be an event set of β. The couple $(\beta, Cutout)$ is a stable branching process iff for every process π of N, one of the two following conditions holds:

1. $\exists C \in Conf(\beta)$: $C \cap Cutout = \varnothing \wedge \Pi(C) = \pi$,
2. $\exists e \in Cutout$, $\exists C \in Conf(\pi)$: $\Pi([e]) = \Pi(C)$.

A simple rule to stop an extension of a cutout event e is to find a noncutout event $\Phi(e)$ such that the cuts of e and $\Phi(e)$ correspond to the same state in N. The final result of the unfolding construction is called an *unfolding* and cutout events are called *cutoff events*.

Definition *(unfolding)*: An unfolding of a net N is a tuple $(\beta, Cutoff, \Phi)$ such that $(\beta, Cutoff)$ is a stable branching process of N and Φ is a mapping from $Cutoff$ to $E(\beta) \backslash Cutoff$ which fulfils $\forall e \in Cutoff$: $h(Cut([e])) = h(Cut([\Phi(e)])$

This simple rule is not adequate to obtain "*good unfoldings*" even for safe P/T net: due to McMillan and presented in [ERV96], a safe P/T net example is proposed for which not all reachable markings are represented in its unfolding. Cutoff rules, as in [McM92] [Esp93] [ERV96] [KKTT96], are defined to insure this reachability property. We consider two rules significant for the detection of illegal behavior: the *size rule* and the *inclusion rule*.

Size rule [McM92]:

1. A cutoff e is associated with an event $\Phi(e)$ such that $[e]$ and $[\Phi(e)]$ lead to the same marking.
2. The size of $[\Phi(e)]$ is strictly smaller than the size of $[e]$.

Inclusion rule [Esp93]:

1. A cutoff e is associated with an event $\Phi(e)$ such that $[e]$ and $[\Phi(e)]$ lead to the same marking.
2. $[\Phi(e)]$ is strictly included in $[e]$.

These rules induce two kinds of unfoldings: size rule unfoldings, and inclusion rule unfoldings. One can note that an inclusion rule unfolding is also a size rule unfolding.

Definition *(size rule unfolding)*: An unfolding $(\beta, Cutoff, \Phi)$ fulfils the size rule iff

$$\forall e \in Cutoff: |[\Phi(e)]| < |[e]|$$

Such an unfolding is called a *size rule unfolding*.

Definition *(inclusion rule unfolding)*: An unfolding $(\beta, Cutoff, \Phi)$ fulfils the inclusion rule iff

$$\forall e \in Cutoff: [\Phi(e)] \subset [e]$$

Such an unfolding is called a *inclusion rule unfolding*.

The unfoldings presented in Figure 6, 7 and 8 are constructed using indiscriminantly the inclusion or the size rules. These rules are adequate to obtain "*good unfoldings*": reachability and deadlock properties of net N can be directly obtained from its unfolding [McM92].

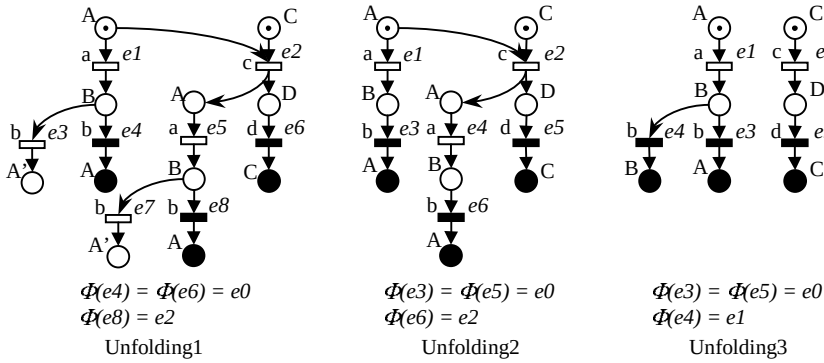


Figure 6: *Unfoldings of PN1, PN2 and PN3*

Proposition *(reachability property)* [McM92]: If *(β, Cutoff, Φ)* is a size rule unfolding of a net *N* then

$$\forall m \in Reach(N), \exists C \in Conf(\beta): C \cap Cutoff = \varnothing \wedge h(Cut(C)) = m.$$

The basic idea of the proof is the following: when a process of *N* crosses a cutoff *e*, the prefix *[e]* can be replaced by *[Φ(e)]* and this substitution leads to the construction of a smallest process leading to the same marking. Hence, when the size or inclusion rules are used, a minimal process (in the number of events) leading to a given marking is necessarily contained in the unfolding.

Rewriting the reachability property for deadlock markings, we obtain the following property.

Proposition *(deadlock property)* [McM92]: Let *(β, Cutoff, Φ)* be a size rule unfolding of a net *N*. A reachable marking *m* of *N* is a deadlock iff $\exists C \in MaxConf(\beta)$ such that $h(Cut(C))=m \wedge C \cap Cutoff = \varnothing$.

A nice algorithm for the detection of deadlocks has been presented by McMillan in [McM92]. This algorithm consists in constructing a configuration containing an event in conflict for each cutoff of the unfolding. For each cutoff *e*, we determine the set *K(e)* of minimal events in conflict with *e*. Then, we search for a conflict-free event set *E* that contains at least one event from each set *K(e)*. If such a set can be defined, then the configuration *[E]* is eventually extended to a maximal configuration which contains no cutoff.

Mortal and deadlock rejects

Existence of mortal and deadlock behavior is simply the direct application of the reachability property.

Proposition *(mortal rejects)***:** Let *(β, Cutoff, Φ)* be a size rule unfolding of a net *N*. *N* can realize an illegal mortal behavior iff $\exists b \in B(\beta)$ such that $h(b) \in Mortal(N)$.

Proposition *(deadlock rejects)***:** Let *(β, Cutoff, Φ)* be a size rule unfolding of a net *N*. *N* can realize an illegal deadlock behavior iff $\exists b \in B$, $\exists C \in MaxConf(\beta)$ such that $h(b) \in Deadlock(N) \wedge b \in Cut(C) \wedge C \cap Cutoff = \varnothing$.

A good algorithm for the detection of deadlock rejects can be found by adapting the deadlock detection algorithm presented by McMillan. For deadlock rejects, we

also have to insure that the configuration contains an event in conflict for each event in the output of the condition *b*. For each cutoff *e* or event *e* in the output of *b*, we determine the set *K(e)* of minimal events in conflict with *e*. Then, we search for a conflict-free event set *E* that contains no event in conflict with *b* and containing at least one event of each set *K(e)*. If such a set can be defined, then the configuration $[E] \cup [b]$ is eventually extended to an illegal deadlock configuration.

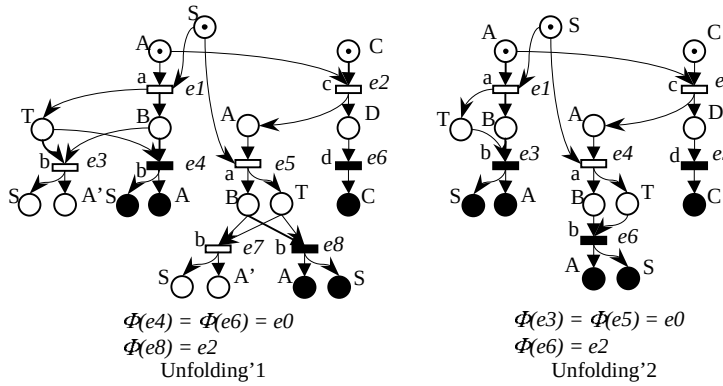


Figure 7: *Analysis of PN1 and PN2*

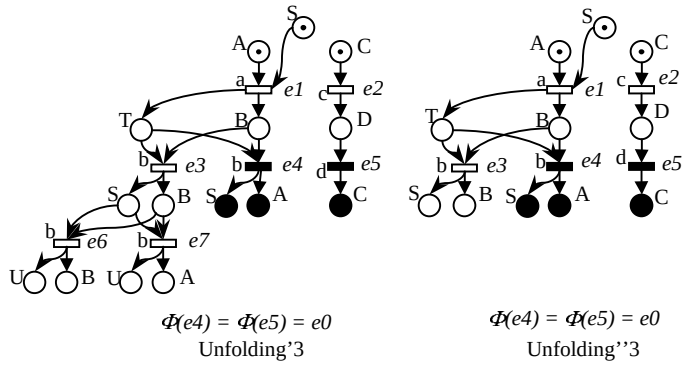


Figure 8: *Analysis of PN3*

To detect the deadlock reject of the unfolding 1 (Figure 6), we construct for each cutoff the set of events in conflict with it: *K(e4)={e2,e3}, K(e6)={e1}, K(e8)={e1,e7}*. From them, we deduce the set *{e1,e3}*. The configuration *[e1,e3]* leads eventually to the deadlock marking *(A', C)*. For the unfolding 2 (Figure 6), we have *K(e3)={e2}, K(e5)=K(e6)={e1}*. As *e1* and *e2* are in conflict, the net is deadlock-free. For the unfolding 3 (Figure 6), the result is immediate: $K(e3)=K(e4)=K(e5)=\varnothing$. The branching processes *unfolding'1, unfolding'2* and *unfolding'3* of Figures 7 and 8 are obtained from the synchronization of the P/T nets of Figure 4 with the tester 2 of Figure 3. From these unfoldings, we can deduce a mortal reject for the net *PN3*: the corresponding unfolding contains a condition labeled *U*.

Live and divergent rejects

The detection of live and divergent rejects relates to the problem of the characterization of infinite behaviors. This problem is difficult: in [Wal98] the author and the referees have been fooled by the apparent simplicity of the problem. In this work, two graphs for which nodes are cutoffs and their images have an important function for the detection of infinite behaviors:

1. Upper-concurrent graph: the cutoffs have as successor their images with respect to Φ and the images have as successor the cutoffs which are upper to or concurrent with them.
2. Upper graph: the cutoffs have as successor their images and the images have as successor the cutoffs which are upper to them.

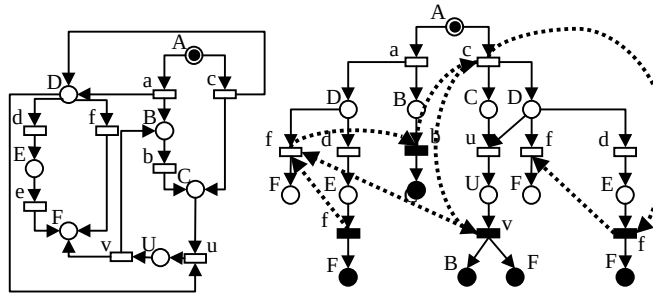


Figure 9: *Erroneous detection of infinite behavior in the upper-concurrent graph*

Reducing an infinite process by the cutoffs it crosses, we construct an infinite path in the upper-concurrent graph. An infinite path in the upper graph induces an infinite behavior passing through the marking associated to the local configuration of the cutoffs explored by the path. Contrary to the allegations presented in [Wal98], any of these graphs allows the characterization of infinite behaviors in a definitive way. The upper-concurrent graph can induce some non-existent infinite behaviors of the P/T net (see counter example in Figure 9); the upper graph does not necessarily detect the infinite behaviors (Figure 10). These graphs must be used only as necessary (upper-concurrent graph) or sufficient (upper-graph) conditions for the existence of illegal live or divergent behaviors.

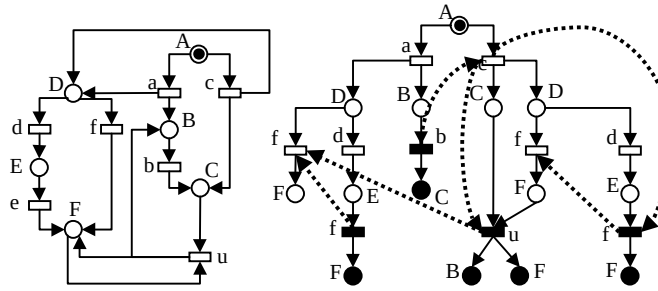


Figure 10: *Non detection of infinite behavior in the upper graph*

When the inclusion rules are used for the unfolding construction, infinite processes looping on a similar event can be reduced using only one cutoff. Then, the detection

of live and divergent rejects is limited to infinite processes that can be reduced to a single cutoff.

Proposition *(cycle property)***:** Let $(\beta, Cutoff, \Phi)$ be an inclusion rule unfolding of a net N. N has an infinite behavior iff *Cutoff* is not empty.

Proof

1) If the event set *Cutoff* is not empty, let e be an event in *Cutoff*. The infinite process $\Pi([\Phi(e)]).\Pi([e]\backslash[\Phi(e)]) \ldots \Pi([e]\backslash[\Phi(e)]) \ldots$ induces an infinite behavior of N.
2) If N has an infinite behavior, we can build a process as big as we want and there then exists a process π which contains more events than the unfolding. Because the unfolding is a stable branching process, this process must cross a cutoff. It proves that event set *Cutoff* is not empty.

As the presence of a cutoff denotes a cycle, a place of live reject represented in an unfolding between a cutoff and its image denotes an illegal live behavior. We demonstrate that this condition is necessary in the case of inclusion rule unfoldings.

Proposition *(live property)***:** Let $(\beta, Cutoff, \Phi)$ be an inclusion rule unfolding of a net N. N can produce an illegal live behavior iff $\exists e \in Cutoff, \exists b \in {}^\bullet([e] \setminus [\Phi(e)])$: $h(b) \in Live(N)$.

Proof

1) If $\exists b \in B(\beta), \exists e \in Cutoff$ such that $h(b) \in Live(N) \wedge b \in {}^\bullet([e] \setminus [\Phi(e)])$. The infinite process $\Pi([\Phi(e)]) \,.\, \Pi([e]\backslash[\Phi(e)]) \ldots \Pi([e]\backslash[\Phi(e)]) \ldots$ induces an illegal live behavior of N.
2) If N can produce an illegal live behavior, there exists processes which contain more live conditions than the unfolding. Let π be one of these processes containing a minimal number of events. Suppose now that

$$\forall e \in Cutoff, \forall b \in {}^\bullet([e] \setminus [\Phi(e)]): h(b) \notin Live(N) \qquad (1)$$

Because the unfolding is a stable branching process, this process must cross a cutoff e: $\pi = \Pi([\Phi(e)]) \,.\, \Pi([e]\backslash[\Phi(e)]) \,.\, \pi'$, where π' is the suffix of π. The new process $\Pi([\Phi(e)]) \,.\, \pi'$ contains as many live conditions as π and fewer events. It proves that the hypothesis (1) is false and that

$$\exists e \in Cutoff, \exists b \in {}^\bullet([e] \setminus [\Phi(e)]): h(b) \in Live(N).$$

The characterization of divergent rejects is a more difficult task. We demonstrate that the presence of particular cutoffs is a necessary and sufficient condition in the case of safe P/T nets. The generalization to bounded nets of this property is still a conjecture.

Proposition *(divergent property)***:** Let $(\beta, Cutoff, \Phi)$ be an inclusion rule unfolding of a safe net N, N can produce an illegal divergent behavior iff $\exists b \in B(\beta), \exists e \in Cutoff$ such that $h(b) \in Divergent(N) \wedge b \,//\, ([e] \setminus [\Phi(e)])$.

Proof

1) Assume $\exists b \in B(\beta), \exists e \in Cutoff$ such that $h(b) \in Divergent(N) \wedge b \,//\, ([e] \setminus [\Phi(e)]$. The infinite process $\Pi([\{b, \Phi(e)\}]) \,.\, \Pi([e]\backslash[\Phi(e)]) \ldots \Pi([e]\backslash[\Phi(e)]) \ldots$ induces an illegal divergent behavior of N.
2) Assume N can produce an illegal divergent behavior, there exists an infinite divergent process π: $\exists b \in B(\pi)$: $h(b) \in Divergent(N) \wedge b^\bullet = \varnothing$. Let π be one of these processes containing a minimal number of events before b ($|[b]|$ is minimal).

Because the unfolding is a stable branching process, this process must cross a cutoff event e: $\pi = \Pi([\Phi(e)]) \,.\, \Pi([e]\setminus[\Phi(e)]) \,.\, \pi'$ (where π' is the suffix of π). The new process $\pi_r = \Pi([\Phi(e)]) \,.\, \pi'$ is still a divergent process ($b\bullet=\emptyset$ in π and then $b\bullet=\emptyset$ in π_r). We just have to prove that if $[b]\cap([e]\setminus[\Phi(e)])\neq\emptyset$ then $|[b]|$ in π_r is less than $|[b]|$ in π. Indeed if $[b]\cap([e]\setminus[\Phi(e)])=\emptyset$ then b and e are the desired condition and event, else π is not *minimal*.

Let prove that

$$h(\{b'\in Cut([e]): b'\leq b\}) \subseteq h(\{b'\in Cut([\Phi(e)]): b'\leq b\}) \qquad (1)$$

Let us decompose $Cut([\Phi(e)])$:

$$Cut([\Phi(e)]) = m1 \cup m1' \text{ with } m1\leq b \text{ and } m1'\,//\,b. \qquad (2)$$

We can note that $m1 = \{b'\in Cut([\Phi(e)]): b'\leq b\}$ and $m1' = Cut([\Phi(e)])\setminus m1$.

Let $m = Cut([\Phi(e)]\cup([e]\cap[b]))$. We can note that $[\Phi(e)]\cup([e]\cap[b])$ is a configuration because $[\Phi(e)]\subseteq[e]$ and so the new events added to $[\Phi(e)]$ cannot be in conflict. m can be obtained from $Cut([\Phi(e)])$ by firing the events of $([e]\setminus[\Phi(e)])\cap[b]$. Because $m'\,//\,b$ then $m1'\subseteq m$. So m can be decomposed into

$$m = m2 \cup m2' \cup m'1 \text{ with } m2\leq b \text{ and } m2'\leq b. \qquad (3)$$

Let us decompose $Cut([e])$:

$$Cut([e]) = m3 \cup m3' \text{ with } m3\leq b \text{ and } m3'\,//\,b. \qquad (4)$$

We can note that $m3 = \{b'\in Cut([e]): b'\leq b\}$ and $m3' = Cut([e])\setminus m$. Because $Cut([e])$ can be obtained from m by firing the events of $([e]\setminus[\Phi(e)])\cap[b]$, we deduce that

$$m2 = m3. \qquad (5)$$

Using the cutoff property $(h(Cut([\Phi(e)])=h(Cut([e]))$, we have

$$h(m3) \subseteq h(m1) \cup h(m1'). \qquad (6)$$

Applying the fact that the net is safe to the marking m in (3), we have

$$h(m2) \cap h(m1') = \emptyset. \qquad (7)$$

Using the equality $m2=m3$ (5) and formulas (6) and (7), we induce that

$$h(m3) \subseteq h(m1)$$

which is exactly formula (1).

The formula (1) proved in the first step implies that the events of $[b]$ in process π_r come from events of $[b]\setminus[Cut([e])$ or $[b]\setminus Cut([\Phi(e)]$ in process π. This proves that $|[b]|$ in process π_r is less than $|[b]|$ in process π. Moreover if $[b]\cap([e]\setminus[\Phi(e)])\neq\emptyset$, they have different sizes.

The unfolding of PN3 (Figure 4) synchronized with the tester 3 (Figure 3) is the branching process *unfolding''3* (Figure 8.) In this unfolding, a divergent reject is detected: the events *{e2,e5}* are before the cutoff *e5* and after its image *e0*, and are concurrent to the divergent condition *T*.

4. Unfolding Graph

In this section, we introduce unfolding graphs and present their usefulness for the detection of illegal behaviors. Moreover, we show how to construct failure transition systems that are failure equivalent to a labeled P/T.

Elementary unfolding graph properties

The construction of an unfolding graph begins by unfolding the net *N*. At each state, we obtain a stable branching process *(β, Cutout)*. A simple rule to stop an

extension of cutout e is to use the size rule, another rule is to decide to build a new branching process from state $Cut([e])$. Such a last cutout is called a *bridge*. By iterating this construction, we obtain a graph of stable branching processes. Each node is called a *partial size rule unfolding* and the resulting graph an *unfolding graph*.

Definition *(partial size rule unfolding)*: A partial size rule unfolding of a net N is a tuple $(\beta, Bridge, Cutoff, \Phi)$ such that $(\beta, Bridge \cup Cutoff)$ is a stable branching process of N and Φ is a mapping from $Cutoff$ to $E(\beta) \setminus (Bridge \cup Cutoff)$ which fulfils

$$\forall e \in Cutoff: h(Cut([e])) = h(Cut([\Phi(e)]) \wedge |[\Phi(e)]| < |[e]|.$$

Definition *(unfolding graph)*: An unfolding graph of a net N is a graph (G,A, g_{init}) where

- $\forall g \in G$: g is a partial size rule unfolding of net $(N, Min(g))$ (N with a new initial marking $Min(g)$) where $Min(g) \in Reach(N)$,
- $\forall g \in G, \forall e \in Bridge(g), \exists! g' \in G: (g, e, g') \in A \wedge h(g)(Cut([e])) = h(g')(Min(g'))$,
- $h(g_{init})(Min(g_{init})) = m_0$, and
- $\forall g \in G$, there exists a path in (G,A, g_{init}) from g_{init} to g.

Unfolding graphs have the same *good properties* as the size rule unfolding: every reachable marking of a net N is represented in its unfolding graph. This property is based on the fact that each process from the initial marking of the net is represented in the graph. We start by reducing the process from the initial node of the graph. When a process crosses a cutoff e, the prefix $[e]$ can be replaced by $[\Phi(e)]$ as for unfoldings. When a process crosses a bridge e, the prefix $[e]$ is deleted from the process. The search must be continued in the successor node corresponding to the crossed bridge. It is clear that in this last case, the transformed searched process has a initial marking equal to the cut associated to the crossed bridge (see [Poi96]).

Proposition *(reachability property):* Let (G,A, g_{init}) be an unfolding graph of a net N. m is a reachable marking of N iff

$$\exists g \in G, \exists C \in Conf(g): \quad C \cap (Bridge(g) \cup Cutoff(g)) = \emptyset \wedge h(g)(Cut(C)) = m.$$

Rewriting the reachability property for deadlock markings, we obtain the following property.

Proposition *(deadlock property):* Let (G,A, g_{init}) be an unfolding graph of a net N. A reachable marking m of N is a deadlock iff

$$\exists g \in G, \exists C \in MaxConf(g): h(g)(Cut(C)) = m \wedge C \cap (Bridge \cup Cutoff) = \emptyset.$$

The efficient algorithm [McM92] for the detection of deadlocks is still applicable for unfolding graphs: it suffices to apply it to each node.

Mortal and deadlock rejects

Existence of mortal and deadlock behavior is simply the direct application of the reachability property.

Proposition *(mortal rejects)***:** Let (G,A, g_{init}) be an unfolding graph of a net N. N can produce an illegal mortal behavior deadlock iff

$$\exists g \in G, \exists b \in B(g): h(g)(b) \in Mortal(N).$$

Proposition *(deadlock rejects)***:** Let (G,A, g_{init}) be an unfolding graph of a net N. N can produce an illegal deadlock behavior iff

$$\exists g \in G, \exists b \in B, \exists C \in MaxConf(g):$$

$$h(g)(b) \in Deadlock(N) \wedge b \in Cut(C) \wedge C \cap (Bridge \cup Cutoff) = \emptyset.$$

For detecting deadlock reject behaviors, we can apply our adaptation of the McMillan algorithm to each node of the unfolding graph.

Live and divergent rejects

To hope to detect infinite behaviors of the net, we need to put more constraints on the cutoff rule. Indeed, in the unfolding study, we have proved that the *size rule* is not a *good rule* for infinite behavior detection (see counter examples). When the inclusion rule is used for the unfolding construction for some node, the detection of infinite live and divergent reject is limited to the structural properties of the unfolding graph. When the inclusion rule is used to construct an unfolding, the node is a partial inclusion rule unfolding.

Definition *(partial inclusion rule unfolding)*: A partial size rule *unfolding* $(\beta, Bridge, Cutoff, \Phi)$ fulfils the inclusion rule iff

$$\forall e \in Cutoff: [\Phi(e)] \subset [e].$$

Such a unfolding is called a *partial inclusion rule unfolding*.

The detection of live reject behaviors is obvious. Such a behavior can be detected locally to a node (like in the unfolding) or globally as a particular cycle in the unfolding graph.

Proposition *(live property)***:** Let (G,A, g_{init}) be an unfolding graph of a net N such that for every g in G:

$(\exists b \in B(g): h(g)(b) \in Live(N)) \Rightarrow g$ is a partial inclusion unfolding.

N can realize an illegal live behavior iff

$$\exists g \in G, \exists e \in Cutoff(g), \exists b \in {}^\bullet([e] \setminus [\Phi(e)]): h(b) \in Live(N) \qquad (1)$$

or

$$\exists \text{ a cycle } (g_i, e_i, g_{i+1})_{i \in [0,n-1]} \text{ in } (G,A, g_{init}): h({}^\bullet[e_0]) \cap Live(N)) \neq \emptyset. \qquad (2)$$

Proof

1. If (1) then $\Pi = \Pi([\Phi(e)]). \Pi([e] \setminus [\Phi(e)]). \Pi([e] \setminus [\Phi(e)])$ … induces an illegal live behavior from accessible marking $h(Min(g))$ (notice that g is a partial inclusion unfolding).
   If (2) then $\Pi = \Pi([e_0])\, \Pi([e_1]) \ldots \Pi([e_{n-1}]) \ldots \Pi([e_0])\, \Pi([e_1]) \ldots \Pi([e_{n-1}])$ induces an illegal live behavior from accessible marking $h(Min(g_0))$.
2. Consider a process Π. If not (1), by cutoff reductions of Π the number of live events does not change. If not (2), a bridge reduction that can change the number of live events cannot be used twice for the reduction of process Π. So if not (1) and not (2), process Π has a limited number of live events (at most the sum of live events of the unfoldings of the graph +1).

The detection of divergent reject behaviors in unfolding graphs is a difficult task even for safe nets. Since such a behavior can be projected in a cycle of the graph, we have to restrain the possible projections.

Proposition *(divergent property)***:** Let (G,A, g_{init}) be an unfolding graph of a net N such that for every g in G:

$(\exists b \in B(g) \setminus Bridge(g)^\bullet: h(g)(b) \in Divergent(N)) \Rightarrow g$ is a partial inclusion unfolding $\wedge \mid \{b \in B(g) \setminus Bridge(g)^\bullet \mid h(g)(b) \in Tester(N)\} \mid = 1$.

N can realize an illegal divergent behavior iff

$\exists g \in G, \exists b \in B(g), \exists e \in Cutoff(g): h(g)(b) \in Divergent(N)$ (1)

$\vee$

$\exists$ a cycle $(g_i, e_i, g_{i+1})_{i \in [0,n-1]}$ in (G,A,g_{init}):

$\exists b \in B(g_0): h(b) \in Divergent(N) \wedge$

$\forall i \in [0,n-1], h(^\bullet[e_i]) \cap Divergent(N) = \emptyset$. (2)

Proof

1. If (1) then $\Pi = \Pi([\Phi(e)]).\ \Pi([e]\backslash[\Phi(e)]).\ \Pi([e]\backslash[\Phi(e)])$ … induces an illegal divergent behavior from accessible marking $h(Min(g))$ (notice that g is a partial inclusion unfolding).
   If (2) then $\Pi = \Pi([e_0])\ \Pi([e_1]) \ldots \Pi([e_{n-1}]) \ldots \Pi([e_0])\ \Pi([e_1]) \ldots \Pi([e_{n-1}])$ induces an illegal divergent behavior from accessible marking $h(Min(g_0))$.
2. Consider an infinite divergent process Π: $\exists b \in E(P): b^\bullet = \emptyset \wedge h(b) \in Divergent(N)$. By cutoff and bridge reduction, process Π stays an infinite divergent process. Write Π as $\Pi([b]).\ \Pi'$ and reduce $\Pi([b])$; we obtain a new divergent process $\Pi_1.\ \Pi'$ such that Π_1 is in an unfolding g_1. The divergent condition b in Π_1 corresponds to the unique divergent condition in g_1 (which is not after a bridge). If we can reduce $\Pi_1.\ \Pi'$ by a cutoff, g_1 fulfils (1). Otherwise we reduce it by a bridge and obtain a new process $\Pi_2.\ \Pi_2'$ such that Π_2 is in an unfolding g_2. Once again, the divergent condition b in Π_2 corresponds to the unique divergent condition in g_2 (which is not after a bridge). We stop this reduction process when the process $\Pi_i.\ \Pi_i'$ crosses a cutoff (g_i fulfils (1)) or when by bridge reduction we go back to an unfolding already visited (cycle in the unfolding graph which fulfils (2)).

Failure equivalent graph

When building an unfolding graph of a labeled P/T net by imposing that the controlled actions are associated only to bridge, we capture almost all the interesting behaviors of a net. To study local divergent behaviors, we use the inclusion rule for unfolding. We will see that such a graph induces a failure transition system that is failure equivalent to the net.

Definition *(strict unfolding graph)*: A *strict unfolding graph* with respect to a transition set *Control* of a labeled net *N* is an unfolding graph (G,A,g_{init}) such that for every *g* in *G*:

- g is a *partial inclusion rule unfolding*, and
- $\forall e \in E(g)\backslash Bridge(g): h(e) \notin Control$.

The strict unfolding graph gives an abstraction of the behavior of the net with regard to the set of controlled actions. To obtain a failure transition system, we have to take into account firstly the infinite behaviors hidden in nodes and secondly potentially blocking behaviors. An infinite behavior local to a node is simply detected by the presence of a cutoff. Blocking behaviors correspond to deadlocks in an unfolding for which the firings of a subset of controlled transitions are not allowed. Formally, a failure transition system is defined as following.

Definition *(failure transition system)*: Let (G,A,g_{init}) be a strict unfolding graph of a labeled net $N = (P, T, \Sigma, W, m_0)$ with respect to a controlled action set *Control*. Its failure transition system is $fsys = (G, \Sigma, F, g_0, Control, div, fail)$ where

- $(g, a, g') \in F \Leftrightarrow \exists (g, e, g') \in A: h(g)(e) = a$,

- $div = \{g \in G \mid Cutoff(g) \neq \emptyset\}$,
- $\forall g \in G,\ fail(g) = \{D \subseteq Control \mid \exists C \in MaxConf(g_D) \wedge C \cap (Cutoff(g_D) \cup Bridge(g_D)) = \emptyset\}$.
  where g_D is the unfolding g for which the event labeled by an element of D are removed.

The failure transition system constructed in the definition is failure equivalent to the net.

Proposition *(failure equivalence)*: Let (G,A, g_{init}) be a strict unfolding graph of a labeled net N with respect to a controlled action set *Control*. Its failure transition system is failure equivalent to N with respect to *Control*.

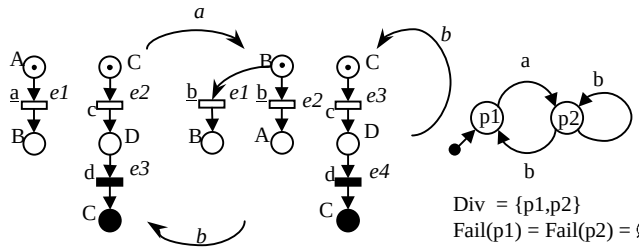


Figure *11*: *Strict unfolding graph and its failure transition system*

The formal proof of this proposition is simple but to long to be shown in this paper. The idea is the following: when synchronizing a tester Z and a failure transition system *fsys*, we almost obtain an unfolding graph. We have simply to replace each node (q, g) of the synchronized system $Z \,//\, fsys$ by the unfolding g for which we properly add the place q. For this unfolding graph, each node is a partial inclusion rule unfolding and contains one place of the tester. One can remark that for each type of illegal behavior detection the characterization previously presented for the unfolding graph is the same as for the failure transition system synchronized with the tester. Figure 11 presents the strict unfolding graph of the net PN3 (Figure 4) and its failure transition system which is failure equivalent to PN3.

5. Experimentation

In this section, unfolding graphs are tested on two classical examples: a specification of a distributed database [Jens86] and the Peterson's algorithm of mutual exclusion for n processes [CP94]. Both examples are scalable, in a such way that the number of reachable states is exponential with respect to a given parameter.

The obtained results are presented in Tables 1 and 2. In the first table we give for each model: the size of the model; the size of its reachability graph; the size of its reduced reachability graph obtained by the stubborn set method [Val91] and the corresponding cpu time; as well as the inclusion rule and size rule unfoldings. The sizes of the unfoldings are denoted by the sum of the numbers of events and conditions. Notice that complete graphs and reduced graphs can be used for the detection of illegal behaviors [Val93]. In the second table, variants of bridge event selection for strict unfolding graphs are tested. Remember that in strict unfolding graphs, the controlled actions can only be associated to bridges. The first tested implementation consists in associating to bridges only controlled actions (*strict unfolding graph without heuristic*). To limit the size of the unfoldings that compose

nodes, other implementations based on three different heuristics have been tested (*strict unfolding graph with heuristic*). Indeed, any event can be chosen as a bridge during the construction without invalidating the construction. For the distributed database, a first heuristic has been used: we impose that in each unfolding there are not two conditions associated to a same place. This heuristic drastically limits the size of the constructed unfoldings. Moreover, it is clear that such an unfolding can not have a cutoff event. For the Peterson's algorithm example with two and three processes, the chosen heuristic imposes that the size of the unfoldings is less that the double of the size of the net. Finally, arbitrary transitions have been designated to be systematically associated with a bridge in Peterson's algorithm with four processes. These different heuristics are used to demonstrate the flexibility of the method.

Model	Model size		Complete graph	Red. graph		Unfolding			
	|P|	|T|	|R|	|R|	time	Inc. rule		Size rule	
						|B|+|E|	time	|B|+|E|	time
BD2	15	8	7	7	0.08	30	0.01	30	0.01
BD4	61	32	109	29	0.16	133	0.03	133	0.01
BD6	139	72	1459	67	0.25	307	0.11	307	0.11
BD8	249	128	17497	121	0.44	553	0.36	553	0.36
Peter2	18	18	50	32	0.11	199	0.05	88	0.01
Peter3	45	57	1065	645	0.88	*	*	4791	37.2
Peter4	84	132	25636	15600	42.31	*	*	*	*

Table 1*: Experimentation results (part 1)*

The controlled actions chosen for the distributed database are *Write(1)* and *Unlock(1)*. This observation allows us to verify that the process 1 necessarily receives the acknowledgment of the others when it asks them to update their local copies. For Peterson's algorithm, the controlled actions are *Ask(1)*, *Enter(1)* and *Leave(1)*. This observation allows us to verify the fairness property which expresses that when the first process asks for an access to the critical section, it eventually obtains it. The two cutoff rules (inclusion and size) are tested in both cases. For each kind of graph, it is given: the size of the graph; the average size of the unfoldings composing the nodes, and the time needed to construct it.

Model	Strict graph without heuristic						Strict graph with heuristic					
	Inc. Rule			Size rule			Inc. rule			Size rule		
	|G|	Av.	time	|G|	Av.	time	|G|	Av.	time	|G|	Av.	time
BD2	2	17	0.00	2	17	0.00	3	13	0.01	3	13	0.01
BD4	2	74	0.03	2	74	0.03	15	36	0.08	15	36	0.08
BD6	2	171	0.11	2	171	0.11	45	67	0.55	45	67	0.55
BD8	2	308	0.33	2	308	0.33	119	103	3.13	119	103	3.13
Peter2	7	39	0.04	7	35	0.04	17	18	0.03	8	31	0.03
Peter3	*	*	*	28	994	182.2	161	48	1.93	161	48	2.00
Peter4	*	*	*	*	*	*	1865	65	118.9	1865	64	108.6

Table 2*: Experimentation results (part 2)*

The tool used for the computation of complete and reduced graphs is Prod [VR92]. For the unfoldings and the equivalent unfolding graphs, we have used our own tool. Note that the unfolder included in Pep [Best96] is more efficient for size rule

unfoldings and that using it for the computation of unfolding graph can improve the time results.

The distributed database model presents a high rate of concurrency and the methods of persistent sets and unfolding take advantage of this. The strict unfolding graph method without heuristic have the same performance in terms of time but graphs have a constant size. The heuristic chosen for the second implementation leads to exponential graph size. We show here that the disadvantages of the first version of unfolding graphs presented in [CP96] have been corrected.

The Peterson's algorithm is highly synchronized and the reduction rate obtained by the persistent set method is then minimized (less than two). For the unfolding technique, one can notice that the size of the resulting net becomes large. The more the size of the unfolding net grows during the construction, the more the cost of adding a new event increases. Indeed, this cost depends on the size of the unfolding and the rate of association of conditions to a same place. Using the size cutoff rule, some better results can be obtained but the verification is then limited to the detection of mortal or deadlock rejects. The same phenomena can be remarked for the strict unfolding graph constructed without heuristic. But the flexibility of the algorithm allows us to find heuristics for which a graph can be efficiently constructed even for the inclusion cutoff rule.

6. Concluding Remarks

For the detection of illegal behaviors, we have presented two techniques: unfoldings and unfolding graphs. It seems *a priori* that unfolding is well-suited to the detection of mortal and deadlock rejects while unfolding graphs is well-suited to the detection of live and divergent rejects. However, our experiments show that the flexibility of unfolding graph algorithm allows us to postpone the limits of the unfolding method. Moreover, two major qualities of the unfolding graph method are that it allows one to compute failure equivalent graphs and that the construction and the analysis of a net synchronized with a tester can be done on-the-fly.

References

[Bes96] Best E., *Partial Order Verification with PEP*, Proc. POMIV'96, G. Holzmann, D. Peled and V. Pratt (eds), Am. Math. Soc., 1996.

[CP94] Couvreur J.M, Paviot-Adet E, *New Structural Invariants for Petri Nets Analysis*, Proc. of ICATPN'94, LNCS 815, p 199-218, 1994

[CP96] Couvreur J.-M., Poitrenaud D., *Model Checking based on Occurrence Net Graph*, Proc. of Formal Description Techniques IX, Theory, Applications and Tools, p. 380-395, 1996.

[Eng91] Engelfriet J., *Branching Processes of Petri Nets*, Acta Informatica, vol. 28, p. 575-591, 1991.

[Esp93] Esparza J., *Model Checking using Net Unfoldings*, Proc. of TAPSOFT'93, LNCS vol. 668, p. 613-628, 1993.

[ERV96] Esparza J., Römer S., Vogler W. *An Improvement of McMillan's Unfolding Algorithm*, Proc. of TACAS'96, LNCS vol. 1055, p. 87-106, 1996.

[God96] Godefroid P., *Partial-Order Methods for the Verification of Concurent Systems*, LNCS 1032, 1996.

[Jens86] Jensen K., *Coloured Petri Nets*, In Petri Nets: Central Model and their Properties, Advances in Petri Nets 1986, Part 1, Brauer W., Reisig W. & Rozenberg G. (eds), Springer Verlag, LNCS vol 254, pp 248-299, Bad Honnef, Germany, September 1986.

[KKTT96] Kondratyev A., Kishinevsky M., Taubin A., Ten S., *A structural Approach for the Analysis of Petri nets by Reducted Unfolding*, Proc. of ICATPN'96, LNCS 1091, p 346-365, 1996

[McM92] McMillan K.L., *Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits*, Proc. of the 4th Conference on Computer Aided Verification, LNCS vol. 663, p. 164-175, 1992.

[MR97] Melzer S., Römer S., *Deadlock Checking Using Net Unfoldings*, Proc. of the 9th Conference on Computer Aided Verification, p. 352-363, 1997.

[Mur89] Murata T., *Petri Nets: Properties, Analysis, and Applications*, Proc. of the IEEE, vol. 77, n°4, p. 541-580, 1989.

[NPW81] Nielsen M., Plotkin G., Winskel G., *Petri Nets, Events Structures and Domains, Part I*, Theoretical Computer Science, vol. 13, n°1, p. 85-108, 1981.

[Pel93] Peled D., *All from One, One for All: on Model Checking Using Representatives*, Proc. on the 5th Conference on Computer Aided Verification, LNCS vol. 697, p. 409-423, 1993.

[Poi96] Poitrenaud D., *Graphes de Processus Arborescents pour la vérification de Propriétés*, Thèse de Doctorat de l'Université Paris IV, 1996.

[Val90] Valmari A., *Compositional State Space Generation*, Proc of ICATPN'90, p. 43-62, Paris, France, 1990.

[Val91] Valmari A., *Stubborn Sets for Reduced State Space Generation*, Advances in Petri Nets, LNCS vol. 483, p. 491-515, 1991.

[Val93] Valmari A., *On-the-fly Verification with Stubborn Sets*, Proc. of the 5th Conference on Computer Aided Verification, LNCS vol. 697, p. 397-408, 1993.

[VR92] Varpaaniemi K., Rauhamaa M., *The Subborn Set Method in Practice*, Advances in Petri Nets, LNCS vol. 616, p. 389-393, 1992.

[Wal98] Wallner F., *Model Checking LTL using Net Unfolding*, Proc. on the 10th Conference on Computer Aided Verification, 1998.

Five Classes of Invariant-Preserving Transformations on Colored Petri Nets*

To-Yat Cheung
Dept. of Computer Science
City University of Hong Kong
Hong Kong, China
E-mail: cscheung@cityu.edu.hk

Yiqin Lu
Dept. of Electronic Engineering
South China University of Technology
Guangzhou 510641, China
E-mail: eeyqlu@scut.edu.cn

Abstract. Transformations on a system specification are often used as a means for simplifying the process of verification. When applying a transformation, it is an important issue whether some specific properties of the system will be preserved or not. For systems specified in colored Petri nets, this paper provides the criteria for determining the preservation of place-invariants and transition-invariants under five classes of very general transformations, namely, Insertion, Elimination, Replacement, Composition, and Decomposition. Applications to flexible manufacturing engineering systems and telecommunications systems are discussed.

1 Introduction

For systems specified in Petri nets, transformations are often used to modify their structures or to simplify the processes of specification, verification and analysis. For both purposes, an important issue is whether some specific properties of the system will be preserved (in certain sense) under the transformation. For basic Petri nets, this issue has been studied extensively [2,8,20]. Recently, two detailed reviews covering many classes of specific and general transformations for preserving such properties as liveness, boundedness, invariants, etc. have been given by Cheung, et al. [7,8].
In the last decade, basic Petri nets have been extended to colored Petri nets (CP-nets) in response to the rapid expansion of system application and complexity in a multi-user or multi-media computing environment. In particular, CP-nets have been extensively applied for specification and verification in such areas as network protocols, ISDN services [16], manufacturing engineering, etc. However, because of the lack of a theoretical foundation and computational processes, the approach of applying property-preserving transformations on CP-nets for system synthesis and analysis has not been progressing as rapidly as desired. At present, as summarised below, only a very limited amount of work has been reported in the literature.

* This research was supported by City University of Hong Kong, Research Grants Council of Hong Kong, National Natural Science Foundation of China and Guangdong Natural Science Foundation.

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 384-403, 1999.
© Springer-Verlag Berlin Heidelberg 1999

A. Transformations Preserving Liveness, Boundedness, etc.
Berthelot's [2] proposed reductions for basic Place/transition nets which preserve liveness, safeness, boundedness, etc. Haddad [12] extended them to CP-nets. However, it is difficult to apply the latter's methods as they are based on the complicated process of unfolding the CP-net. Benalycherif et al. [1] investigated the synchronization of colored FIFO nets and showed that, when two such nets are composed by merging their transitions and adjacent places, liveness can be preserved under a condition relying on a non-constraining relation between them. Lakos [19] studied the conditions under which an abstraction of CP-nets will preserve such properties as adjacency relationships, color consistency, marking-respecting, step-respecting, place flow, etc.

B. Transformations Preserving Invariants of CP-nets
By extending the Gaussian elimination rules for solving a linear system of equations, Jensen [13] presented four types of transformations that simplify the incidence matrix of a CP-net but preserve its P-invariants or T-invariants. Narahari et al. [23] showed that the invariants of the 'union' of two CP-nets can be composed from those of the component nets. Christensen et al. [10] studied the invariant-preserving problem of modular CP-nets composed by fusing the places and transitions of several modules.
Described above are the only two kinds of invariant-preserving transformations we are aware of. Note that they are not general but are either a Gaussian elimination or a composition of the union type. They cannot be used for detecting invariant preservation under transformations arising from more general and diversified applications.
This paper provides the criteria for preserving the invariants of CP-nets under five classes of general transformations, namely, Insertion, Elimination, Replacement, Composition and Decomposition. The computational processes derived do not require knowledge about the semantics of the transformations or structures of the Petri nets. They are a nontrivial generalisation of similar results for basic Petri nets [8]. The rest of this paper is organised as follows. Section 2 reviews the basic concepts of CP-nets. Sections 3 to 5 describe the five classes of general transformations on CP-nets and the criteria for them to preserve invariants. In Section 6, to conclude this paper and point out some directions for research, we briefly describe the potential application of our results to several areas of problems, mentioning our current work on manufacturing engineering systems [9] and telecommunication systems [7,21] as examples.

2 Colored Petri Nets and Their Invariants

Readers are assumed having basic knowledge of CP-nets. This section first presents the required definitions and notations. Some explanation for beginners is given at the end. More details and examples can be found in [13,14,16].

Notation

1. For a variable or expression x, $Type(x)$ denotes the type of x.
2. For an expression Exp, $Var(Exp)$ denotes the set of free variables in Exp.
3. The inner product of two vectors α and β is written as $\alpha * \beta$.
4. $\mathbf{0}$ denotes a zero-function.
5. For two sets S_1 and S_2 where $S_1 \cap S_2 = \varnothing$, $S_1 \cup S_2$ is denoted as $S_1 + S_2$.

Definition 1 (Multi-set, Multi-set Extension of a Function). A *multi-set* over a finite set S is a function $m: S \rightarrow$ (set of non-negative integers). It can be expressed in the form $\sum_{s \in S} m(s)$'s, where $m(s)$' indicates that element s occurs $m(s)$ times in the multi-set. S_{MS} denotes the set of all multi-sets over S. For two finite sets S and R and a function $F: S \rightarrow R_{MS}$, *the multi-set extension of* F is a function $\hat{F} : S_{MS} \rightarrow R_{MS}$ such that $\forall m \in S_{MS}$, $\hat{F}(m) = \sum_{s \in S} m(s) \cdot F(s)$.

Definition 2 (Generalized Matrix-Multiplication). Let Q, Y *and* Z be three non-empty sets, F be an $m \times n$ matrix whose element F_{ij}: $Q_{MS} \rightarrow Y_{MS}$ is a linear function, H be an m-*vector* whose ith element H_i: $Y_{MS} \rightarrow Z_{MS}$ is also a linear function, and V be an n-*vector* whose jth element is $V_j \in Q_{MS}$. The result of the *generalized matrix-multiplication* $H * F$ is an n-vector S whose jth element is a function S_j: $Q_{MS} \rightarrow Z_{MS}$ such that $\forall a \in Q_{MS}$, $S_j(a) = \sum_{i=1}^{m} H_i\left(F_{ij}(a)\right)$, whereas $F * V$ is an m-vector W whose ith element is a function W_i: $Q_{MS} \rightarrow Y_{MS}$ such that $\forall a \in Q_{MS}$, $W_i = \sum_{j=1}^{n} F_{ij}(V_j)$.

Definition 3 (Colored Petri Net). A *Colored Petri net* (CP-net) is a 6-tuple $N = <P, T, A, C, E, G>$, where:

a. P and T are two disjoint sets of nodes called *places* and *transitions*, respectively. A place is often drawn as a circle and a transition as a rectangle.
b. A is a set of *directed arcs* connecting a place and a transition. An arc is denoted as (p,t) or (t,p) with $p \in P$ and $t \in T$.
c. C is a *color function* on P. That is, $\forall p \in P$, $C(p)$ is a set of *token-colors*.
d. $E: A \rightarrow Exps$ is an *arc expression function*. $Exps$ is a set of expressions. $\forall a \in A$, $Type(E(a))$ is $C(p(a))_{MS}$, where $p(a)$ is the source or destination place of a.
e. $G: T \rightarrow Prs$ is a *guard function,* where Prs is a set of first order formulas. (A guard function always evaluated to true may be omitted.)

Hereafter, all relevant notations are referred to a CP-net $N = <P, T, A, C, E, G>$.

Notation

a. For $x \in T \cup P$, $A(x)$ denotes all the arcs in A that are adjacent to x.
b. For $t \in T$, $Var(t)$ denotes the set of variables either in the guard of t or in the expression of any arc adjacent to t, i.e., $Var(t) = \{ v \mid v \in Var(G(t)) \vee \exists a \in A(t): v \in Var(E(a)) \}$.

Definition 4 (Binding). A *binding* b for an expression Exp, in notation $Exp(b)$, is an assignment of values (i.e., tokens of certain colors) to $Var(Exp)$ for the evaluation of Exp. A *binding for a transition* $t \in T$ is a binding of $G(t)$ and every expression associated with an arc in $A(t)$. The set of all bindings for t is denoted as $B(t)$.

Definition 5 (Incidence Matrix). *The incidence matrix* I of N is a $|P| \times |T|$ matrix whose element I_{ij} is a multi-set extension of function $f_{ij}(b) = E(t_j,p_i)(b) - E(p_i,t_j)(b)$, where $t_j \in T$, $p_i \in P$ and $b \in B(t_j)$.

Definition 6 (Weight-Function, Support). A *place weight-function* W_P is a function on P such that $\forall p \in P$, $W_P(p)$ is a mapping from $C(p)_{MS}$ to X_{MS}, where X is a non-

empty set. The *support* of W_P is $\{p \in P \mid W_P(p) \neq \mathbf{0}\}$. W_P is often written as a $|P|$-vector with $W_P(p_i)$ as its ith element. A *transition weight-function* W_T is a function on T such that $\forall t \in T$, $W_T(t) \in B(t)_{MS}$. The *support* of W_T is $\{ t \in T \mid W_T(t) \neq \mathbf{0}\}$. W_T is often written as a $|T|$-vector with $W_T(t_i)$ as its ith element.

Notation. A place/transition weight-function W_P/W_T with support $\{p_{m_1}, p_{m_2}, \ldots, p_{m_k}\} / \{t_{m_1}, t_{m_2}, \ldots, t_{m_k}\}$ may be denoted as:

$$\begin{pmatrix} p_{m_1} & p_{m_2} & \cdots & p_{m_k} \\ W_P(p_{m_1}) & W_P(p_{m_2}) & \ldots & W_P(p_{m_k}) \end{pmatrix} \Big/ \begin{pmatrix} t_{m_1} & t_{m_2} & \cdots & t_{m_k} \\ W_T(t_{m_1}) & W_T(t_{m_2}) & \ldots & W_T(t_{m_k}) \end{pmatrix}.$$

Definition 7 (Invariant). Let I be the incidence matrix. A place weight-function W_P is said to be a *place-invariant* (*P-invariant*) iff $W_P * I = \mathbf{0}$. A transition weight-function W_T is said to be a *transition-invariant* (*T-invariant*) iff $I * W_T = \mathbf{0}$.

Definition 8 (Weighted Net Flow). Let I be the incidence matrix, W_P be a place weight-function and W_T be a transition weight-function. For $p_i \in P$, $T' \subseteq T$, $t_j \in T$ and $P' \subseteq P$, the *W_T-weighted net flow* of p_i with respect to T' is defined as $NF(p_i, T', W_T) = \sum_{t_k \in T'} I_{i,k}(W_T(t_k))$ and the *W_P-weighted net flow* of t with respect to P' is defined as $NF(t_j, P', W_P) = \sum_{p_k \in P'} W_P(p_k) \circ I_{k,j}$, where ° is a functional composition.

The above concepts and definitions for CP-nets are all generalizations from basic Petri nets where only one kind of tokens is involved. A CP-net deals with many kinds (called *token-colors* or just *colors*) of tokens. Each of its places, transitions and arcs may accommodate only specific subsets of the colors and number of tokens of each color. Note that, in general, a designer may use his/her own interpretations of these concepts during application. For example, the color function $C(p)$ in Definition 3 imposes a restriction on the colors and the maximum number of tokens of each color allowed to be deposited into place p. Then, if in Definiton 1 we let S be P (i.e., the set of places) and R be the set of colors, we have $C: P \rightarrow R_{MS}$. Also, the arc expression function $E(a)$ in Definition 3 states the multi-set of colors either required from p in order to fire t (in case $a = (p,t)$) or to be deposited into p after firing t (in case $a = (t,p)$). In either case, it is a multi-set extension of the form: E: $C(p(a))_{MS} \bullet R_{MS}$. (Compare Definitions 1 and 2.) Furthermore, at different times, tokens of different subsets of colors may be required in order to fire a transition. Hence, variables within scopes are often used to express such variations and limitation. They are *bound* when all their variables 'pick up' the *actual* tokens at run time. Lastly, the place/transition weight-function is a generalization of the marking/firing vector of basic Petri nets, where a support includes the non-zero elements.

3 Insertion T-IN and Elimination T-EL

T-IN/T-EL adds/deletes a set of places, transitions and arcs to/from a CP-net.

Example. Fig. 1 shows an insertion T-IN, where N_A is transformed into N'_A by inserting the set of places $IP = \{p_4, p_5, p_6\}$, the set of transitions $IT = \{t_4\}$ and the set

of arcs $IA = \{(p_3,t_4), (p_4,t_4), (t_4,p_4), (t_4,p_5), (t_4,p_6), (p_5,t_3), (p_6,t_3)\}$ between (p_3,t_3) and t_3. A color function IC assigns a color set to each of the inserted places in IP, e.g., $IC(p_4) = D$, $IC(p_5) = IC(p_6) = D \times D$. A guard function IG assigns a guard expression to each of the inserted transitions in IT, e.g., $IG(t_4) = true$. An arc expression function IE assigns an expression to each of the inserted arcs in IA, e.g., $IE(p_3,t_4) = 3‘(x,y)$, $IE(p_4,t_4) = z$, $IE(t_4,p_3) = z$, $IE(t_4,p_5) = 3‘(x,z)$, $IE(t_4,p_6) = 3‘(z,y)$, $IE(p_5,t_3) = (x,z)$ and $IE(p_6,t_3) = (z,y)$.

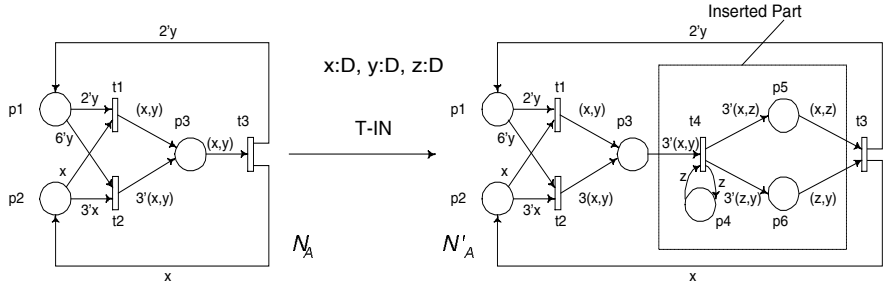
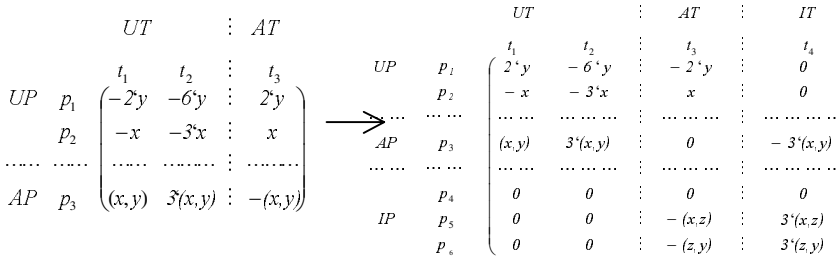


Fig. 1. Insertion T-IN transforms N_A to N'_A by inserting the subnet within the dashed square between arc (p_3,t_3) and t_3 of N.

We divide P/T of N_A into two disjoint sets: AP/AT (i.e., the set of *affected places/transitions* connected to at least one inserted arc) and UP/UT (i.e., the set of *unaffected places/transitions* not connected to any inserted arcs). In Fig. 1, $AP = \{p_3\}$, $UP = \{p_1,p_2\}$, $AT = \{t_3\}$ and $UT = \{t_1,t_2\}$. AA (i.e., the set of *affected arcs* between a place in AP and a transition in AT) is $\{(p_3,t_3)\}$ and is removed in the transformation.



Incidence matrix of N_A Incidence matrix of N'_A

UP/UT, AP/AT, IP/IT: unaffected, affected, inserted places/transitions

Fig. 2. Incidence matrix representation of the transformation in Figure 1.

Formal Description of Insertion T-IN

Insertion T-IN transforms a CP-net $N = <P, T, A, C, E, G>$ to another CP-net $N' = <P', T', A', C', E', G'>$ by inserting a set of places IP, a set of transitions IT and a set of arcs IA. Associated with T-IN are a color function IC defined on IP, an arc-

expression function $IE: IA \rightarrow Exps$ and a guard function $IG: IT \rightarrow Prs$ for assigning functions or expressions to the inserted places IP, transitions IT and arcs IA.

N and N' are related in the following way:

a. $P' \equiv P + IP$, $T' \equiv T + IT$ and $A' \equiv A + IA - AA$, where
$AA = \{ (p, t) \mid p \in AP \wedge p \in AT \} + \{ (t, p) \mid p \in AP \wedge p \in AT \}$, where
$AP = \{ p \mid A(p) \cap IA \neq \emptyset \}$ and $AT = \{ t \mid A(t) \cap IA \neq \emptyset \}$.

b. $C'(p) \equiv IC(p)$ if $p \in IP$
$\equiv C(p)$ if $p \in P$

c. $E'(a) \equiv IE(a)$ if $a \in IA$
$\equiv E(a)$ if $a \in A - AA$

d. $G'(t) \equiv G(t)$ if $t \in T$
$\equiv IG(t)$ if $t \in IT$

3.1 Preservation of P-Invariants under Insertion T-IN

Theorem 1 below states that a place-invariant of a CP-net can be preserved if and only if, after the transformation, the weighted net flow of each of the affected transitions is not changed and the weighted net flow of each of the inserted transitions is $\boldsymbol{0}$.

Theorem 1. Let N be transformed to N' by Insertion T-IN and let $U \subseteq P - AP$, $\Lambda \subseteq AP$ and $\Gamma \subseteq IP$. Suppose W_P is a place weight-function of N with support $U + \Lambda$ and W'_P is a place weight-function of N' with support $U + \Lambda + \Gamma$. If the following conditions:

a. $\forall\, p \in U + \Lambda,\ W_P(p) = W'_P(p)$, (3.1)

b. $\forall\, t \in AT,\ NF(t, \Lambda+\Gamma, W'_P) = NF(t, \Lambda, W_P)$ *and* (3.2)

c. $\forall\, t \in IT,\ NF(t, \Lambda+\Gamma, W'_P) = \boldsymbol{0}$ (3.3)

are satisfied, then W_P is a P-invariant of N iff W'_P is a P-invariant of N'.

Proof: Let UP denote P-AP and UT denote T - AT. Suppose $|T| = n$ and $|UT| = i$, then $|AT| = n - i$. For T-IN, the incidence matrix I is changed to I' as shown below.

$$\begin{array}{c} \\ UP \\ AP \end{array}\!\!\begin{array}{c} \begin{array}{cc} UT & AT \end{array} \\ \begin{pmatrix} X_1 \ldots X_i & X_{i+1} \ldots X_n \\ Y_1 \ldots Y_i & Y_{i+1} \ldots Y_n \end{pmatrix} \end{array} \longrightarrow \begin{array}{c} \\ UP \\ AP \\ IP \end{array}\!\!\begin{array}{c} \begin{array}{ccc} UT & AT & IT \end{array} \\ \begin{pmatrix} X_1 \ldots X_i & X'_{i+1} \ldots X'_n & 0_{n+1} \ldots 0_{n+|IT|} \\ Y_1 \ldots Y_i & Y'_{i+1} \ldots Y'_n & Y_{n+1} \ldots Y_{n+|IT|} \\ 0_1 \ldots 0_i & Z_{i+1} \ldots Z_n & Z_{n+1} \ldots Z_{n+|IT|} \end{pmatrix} \end{array}$$

The incidence matrix I of N Incidence matrix I' of N'

UP/UT, AP/AT, IP/IT: unaffected, affected, inserted place/transitions

Fig. 3. Incidence matrix representation of Insertion (T-IN) in vertical vector format.

In Fig. 3, X_k ($k = 1, \ldots, n$) and X'_k ($k = i+1, \ldots, n$) are vertical $|UP|$-vectors. Y_k ($k = 1,\ldots, n+|IT|$) and Y'_k ($k = i+1, \ldots, n$) are vertical $|AP|$-vectors. Z_k ($k = i+1,\ldots,n+|IT|$) is a vertical $|IP|$-vector. 0_j ($j = 1,2,\ldots, i$) and 0_k ($k = n+1, \ldots, n+|IT|$) are zero $|IP|$-vector and zero $|UP|$-vector, respectively. X_k and X'_k ($k = i+1, \ldots, n$) have the following relationship: For $j = 1,\ldots, |UP|$, $(X_k)_j$ is the multi-set extension of function $f_{jk}(b) = E(t_k,p_j)(b) - E(p_j,t_k)(b)$, where b is a binding of t_k in N, and $(X'_k)_j$ is a mutli-set extension of function $f'_{jk}(b') = E'(t_k,p_j)(b) - E'(p_j,t_k)(b')$, where b' is a binding of t_k in N'. By the definition of T-IN, for $t_k \in AT$ and $p_j \in UP$, $E'(t_k,p_j) = E(t_k,p_j)$ and $E'(p_j,t_k) = E(p_j,t_k)$.

Hence, f_{jk} and f'_{jk} have the same form. However, since $A(t_k)$ will change after the transformation, causing the change of $Var(t_k)$, the domains of f_{jk} and f'_{jk} may be different. Let V_k and V_k' denote $Var(t_k)$ in N and N', respectively. If b and b' satisfy the condition $\forall v \in V_k \cap V_k'$: $b(v) = b'(v)$, then $f_{jk}(b) = f'_{jk}(b')$.

W_P can be expressed as $(\alpha\ \beta)$, where α is a $|UP|$-vector such that $U = \{ p \in UP \mid |\alpha(p)| \neq \mathbf{0} \}$ and β is a $|AP|$-vector such that $\Lambda = \{ p \in AP \mid |\beta(p)| \neq \mathbf{0} \}$. Based on (3.1), W'_P can be expressed as $(\alpha\ \beta\ \gamma)$, where γ is an $|IP|$-vector such that $\Gamma = \{ p \in IP \mid |\gamma(p)| \neq \mathbf{0} \}$. Next, we have to prove that, if W_P and W'_P satisfy (3.2) and (3.3), then the following formula is true:

$$W_P * I = \mathbf{0} \text{ iff } W'_P * I' = \mathbf{0}. \tag{3.4}$$

a. For $k = 1, 2, \ldots, i$, it is obvious that $(\alpha\ \beta) * \begin{pmatrix} X_k \\ Y_k \end{pmatrix} = (\alpha\ \beta\ \gamma) * \begin{pmatrix} X_k \\ Y_k \\ 0_k \end{pmatrix}$.

Hence, $(\alpha\ \beta) * \begin{pmatrix} X_k \\ Y_k \end{pmatrix} = \mathbf{0}$ *iff* $(\alpha\ \beta\ \gamma) * \begin{pmatrix} X_k \\ Y_k \\ 0_k \end{pmatrix} = \mathbf{0}$

b. For $k = i+1, \ldots, n$, by Definition 8, we have $NF(t_k, \Lambda, W_P) = \beta * Y_k$ and $NF(t_k, \Lambda+\Gamma, W'_P) = \beta * Y'_k + \gamma * Z_k$. Together with (3.2), we have $\beta * Y_k = \beta * Y'_k + \gamma * Z_k$. From the analysis of the relationship between X_k and X'_k, we have $\alpha * X_k = \mathbf{0}$ iff $\alpha * X'_k = \mathbf{0}$. Hence, $\alpha * X_k + \beta * Y_k = \mathbf{0}$ iff $\alpha * X'_k + \beta * Y'_k + \gamma * Z_k = \mathbf{0}$.

Hence, we have $(\alpha\ \beta) * \begin{pmatrix} X_k \\ Y_k \end{pmatrix} = \mathbf{0}$ *iff* $(\alpha\ \beta\ \gamma) * \begin{pmatrix} X'_k \\ Y'_k \\ Z_k \end{pmatrix} = \mathbf{0}$

c. For $k = n+1, \ldots, n + |IT|$, by Definition 8, $NF(t_k, \Lambda+\Gamma, W'_P) = \beta * Y_k + \gamma * Z_k$. Together with (3.3), we have $\beta * Y_k + \gamma * Z_k = \mathbf{0}$ and $(\alpha\ \beta\ \gamma) * \begin{pmatrix} 0_k \\ Y_k \\ Z_k \end{pmatrix} = \mathbf{0}$

Hence, if Conditions (3.2) and (3.3) are satisfied, (3.4) is true. □

Corollary 1. Let Insertion T-IN transform N to N', $U = (P - AP) \cap INV$ and $\Lambda = AP \cap INV$.

a. Suppose W_P is a P-invariant of N with support INV and W'_P is a place weight-function of N' with support $INV + \Gamma$, where $\Gamma \subseteq IP$. If U, Λ and Γ satisfy (3.1) - (3.3), then W'_P is a P-invariant of N'.

b. Suppose W'_P is a P-invariant of N' with support INV and W_P is a place weight-function of N. Let $\Gamma = IP \cap INV$. If U, Λ and Γ satisfy (3.1) - (3.3), then W_P is a P-invariant of N.

Example. As a special case of Theorem 1, Corollary 1 makes it easier to find a P-invariant of the transformed/original net through the original/transformed net. Consider Fig. 1 as an example. N_A has a P-invariant $W_P = \begin{pmatrix} p_2 & p_3 \\ x & x \end{pmatrix}$ with support $INV = \{p_2, p_3\}$. Let $U = (P - AP) \cap INV = \{p_2\}$, $\Lambda = AP \cap INV = \{p_3\}$ and $\Gamma = \{p_5\}$. N'_A

has a place weight-function $W'_{P1} = \begin{pmatrix} p_2 & p_3 & p_5 \\ x & x & x \end{pmatrix}$. By Corollary 1.a, since U, Λ, Γ satisfy (3.1) - (3.3), W'_{P1} is a P-invariant of N'_A. Similarly, other place weight-functions $W'_{P2} = \begin{pmatrix} p_2 & p_3 & p_4 & p_5 \\ x & x & z & x \end{pmatrix}$, $W'_{P3} = \begin{pmatrix} p_1 & p_2 & p_3 & p_5 & p_6 \\ y & x & x+2y & x & 2y \end{pmatrix}$ and $W'_{P4} = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ y & x & x+2y & z & x & 2y \end{pmatrix}$ are also P-invariants of N'_A. W_P can also be determined through W'_{P1}, W'_{P2}, W'_{P3} or W'_{P4} by applying Corollary 1.b.

3.2 Preservation of T-Invariants under Insertion (T-IN)

This section considers the duals of Theorem 1 and Corollary 1.

Theorem 2. Let N be transformed to N' by T-IN and $U \subseteq T\text{-}AT$, $\Lambda \subseteq AT$ and $\Gamma \subseteq IT$. Suppose W_T is a transition weight-function of N with support $U + \Lambda$ and W'_T is a transition weight-function of N' with support $U + \Lambda + \Gamma$. If the following conditions:

a. $\forall t \in U$, $W_T(t) = W'_T(t)$, (3.5)

$\forall t \in \Lambda$, $\forall v \in V \cap V'$, $W_T(t)(v) = W'_T(t)(v)$, (3.6)

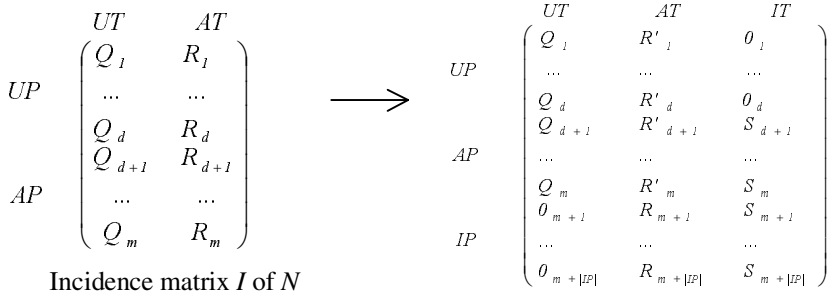
where V denotes $Var(t)$ in N and V' denotes $Var(t)$ in N',

b. $\forall p \in AP$, $NF(p, \Lambda+\Gamma, W'_T) = NF(p, \Lambda, W_T)$ and (3.7)

c. $\forall p \in IP$, $NF(p, \Lambda+\Gamma, W'_T) = \mathbf{0}$ (3.8)

are satisfied, then W_T is a T-invariant of N iff W'_T is a T-invariant of N'.

Proof: Let $UP = P - AP$, $UT = T - AT$, $|P| = m$ and $|UP| = d$. Then, $|AP| = m - d$. By the definition of T-IN, the incidence matrix I is changed to I' as shown in Fig. 4.



Incidence matrix I of N

Incidence matrix I' of N'

UP/UT, AP/AT, IP/IT: unaffected, affected, inserted place/transitions

Fig. 4. Incidence matrix representation of Insertion (T-IN) in horizontal vector format.

In Fig. 4, Q_k ($k = 1,\ldots, m$) is a horizontal $|UT|$-vector. R_k ($k = 1,\ldots, m+|IP|$) and R'_k ($k = 1,\ldots, m$) are horizontal $|AT|$-vectors. S_k ($k = d+1,\ldots, m+|IP|$) is a horizontal $|IT|$-vector. 0_j ($j = 1,\ldots, d$) and 0_k ($k = m+1,\ldots,m+|IP|$) are zero $|IT|$-vector and zero $|UT|$-vector respectively. For $k = 1,\ldots,d$, R_k and R'_k have the following relationship: $(R_k)_j$ ($j = 1,\ldots, |AT|$) is the multi-set extension of function $f_{k,|UT|+j}(b) = E(t_{|UT|+j},p_k)(b) - E(p_k,t_{|UT|+j})(b)$, where b must be a binding of $t_{|UT|+j}$ in N; whereas $(R'_k)_j$ is the mutli-set extension of function $f'_{k,|UT|+j}(b') = E'(t_{|UT|+j},p_k)(b') - E'(p_k,t_{|UT|+j})(b')$, where b' must be a binding of $t_{|UT|+j}$ in N'. From the definition of T-IN, for $t_{|UT|+j} \in AT$ and $p_k \in UP$, $E'(t_{|UT|+j},p_k) = E(t_{|UT|+j},p_k)$ and $E'(p_k,t_{|UT|+j}) = E(p_k,t_{|UT|+j})$. Thus, $f_{k,|UT|+j}$ and $f'_{k,|UT|+j}$ have the

same form. However, because $A(t_{|UT|+j})$ will change after the transformation, $Var(t_{|UT|+j})$ will also change. Hence, $f_{k,|UT|+j}$ and $f'_{k,|UT|+j}$ may have different domains. Let set V_j and set V_j' denote $Var(t_{|UT|+j})$ in N and N' respectively. If b and b' satisfy the condition that $\forall v \in V_j \cap V_j'$: $b(v) = b'(v)$, then $f_{k,|UT|+j}(b) = f'_{k,|UT|+j}(b')$.

Next, we express W_T in the form $(\alpha\ \beta)$, where α is a $|UT|$-vector such that $\{t \in UT \mid \alpha(t) \neq \mathbf{0}\} = U$ and β is an $|AT|$-vector such that $\{t \in AT \mid |\beta(t)| \neq 0\} = \Lambda$. From (3.5) and (3.6), W'_T can be expressed in the form of $(\alpha\ \beta'\ \gamma)$, where β' is the modification of β based on (3.6), and γ is a $|IT|$-vector such that $\{t \in IT \mid |\gamma(t)| \neq 0\} = \Gamma$. Next, we will show that the following formula is true if W_P and W'_P satisfy (3.7) and (3.8).

$$I * W_T = \mathbf{0} \text{ iff } I' * W'_T = \mathbf{0}. \tag{3.9}$$

a. For $k = 1, 2, \ldots, d$, from (3.5) and the relationship of R_k and R'_k, we have $R_k * \beta = \mathbf{0}$ iff $R'_k * \beta' = \mathbf{0}$, and $(Q_k R_k) * \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \mathbf{0}$ *iff* $(Q_k R'_k 0_k) * \begin{pmatrix}\alpha\\ \beta'\\ \gamma\end{pmatrix} = \mathbf{0}$

b. For $k = d+1, \ldots, m$, from Definition 8, we have $NF(p_k, \Lambda, W_T) = R_k * \beta$ and $NF(p_k, \Lambda+\Gamma, W'_T) = R'_k * \beta' + S_k * \gamma$. Together with (3.7), we have $R_k * \beta = R'_k * \beta' + S_k * \gamma$. Hence,

$$(Q_k R_k) * \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = (Q_k R'_k S_k) * \begin{pmatrix}\alpha\\ \beta'\\ \gamma\end{pmatrix} \text{ and } (Q_k R_k) * \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \mathbf{0} \text{ iff } (Q_k R'_k S_k) * \begin{pmatrix}\alpha\\ \beta'\\ \gamma\end{pmatrix} = \mathbf{0}.$$

c. For $k = m+1, \ldots, m + |IP|$, from Definition 8, we have $NF(p_k, \Lambda + \Gamma, W'_T) = R_k * \beta' + S_k * \gamma$. Together with (3.8), we have $R_k * \beta' + S_k * \gamma = \mathbf{0}$. Hence,

$$(0_k R_k S_k) * \begin{pmatrix}\alpha\\ \beta'\\ \gamma\end{pmatrix} = 0$$

Therefore, if Conditions (3.7) and (3.8) are satisfied, (3.9) is true. □

Corollary 2. Let N be transformed to N' by Insertion T-IN, $U = (T - AT) \cap INV$, $\Lambda = AT \cap INV$ and $\Gamma = IT \cap INV$.

a. Suppose W_T is a T-invariant of N with support INV and W'_T is a transition weight-function of N' with support $INV + \Gamma$, where $\Gamma \subseteq IT$. If U, Λ, and Γ satisfy (3.5) - (3.8), then W'_T is a T-invariant of N'.

b. Suppose W'_T is a T-invariant of N' with support INV and W_T is a transition weight-function of N with support $U + \Lambda$. If U, Λ and Γ satisfy (3.5) - (3.8), then W_T is a T-invariant of N.

Example. As a special case of Theorem 2, Corollary 2 can be applied to find T-invariants of the transformed/original net from the original/transformed net. For example, in Fig. 1, $W_{T1} = \begin{pmatrix} t_2 & t_3 \\ <x=a, y=b> & 3`<x=a, y=b> \end{pmatrix}$ and $W_{T2} = \begin{pmatrix} t_1 & t_3 \\ 3`<x=a, y=b> & 3`<x=a, y=b> \end{pmatrix}$ are two $T$-invariants of $N_A$. The support of $W_{T1}$ is $INV = \{t_2, t_3\}$. Let $U = (T - AT) \cap INV = \{t_2\}$, $\Lambda = AT \cap INV = \{t_3\}$ and $\Gamma = \{t_4\}$. $W'_{T1} = \begin{pmatrix} t_2 & t_3 & t_4 \\ <x=a, y=b> & 3`<x=a, y=b, z=c> & <x=a, y=b, z=c> \end{pmatrix}$ is a transition weight-function of N'_A. By Corollary 2.a, since U, Λ and Γ satisfy (3.5) - (3.8), W'_{T1} is

the T-invariant of N'_A. Similarly, from W_{T2}, we can show that $W'_{T2} = \begin{pmatrix} t_1 & t_3 & t_4 \\ 3`<x=a, y=b> & 3`<x=a, y=b> & <x=a, y=b, z=c> \end{pmatrix}$ is a T-invariant of N'_A. Conversely, given W'_{T1} and W'_{T2}, we can find W_{T1} and W_{T2} by Corollary 2.b.

3.3 Elimination T-EL

Elimination T-EL reverses the roles of N and N' in Insertion T-IN. The conditions for preserving invariants are the same as those given in Theorem 1 and Theorem 2. Some reductions of CP-nets [12] can be seen as special cases of Elimination.

4 Replacement T-RE

Replacement may be divided into two cases: *Replacement-of-Places* T-RP replaces a set of places and some adjacent arcs with other places and arcs. *Replacement-of-Transitions* T-RT is similar to T-RP except that it deals with transitions.

Example. Fig. 5 shows that N_B is transformed to N'_B by replacing the set of places AP = {p_4, p_5, p_6} with RP = {p'_4, p'_5} and the set of adjacent arcs AR = {(t_3,p_4), (t_3,p_5), (t_3,p_6) (t_4,p_4), (p_4,t_5), (p_5,t_5), (p_5,t_6), (p_6,t_6)} with RA = {(t_3,p'_4), (t_3,p'_5), (t_4,p'_4), (p'_4,t_5), (p'_4,t_6), (p'_5,t_5), (p'_5,t_6)}. The color function RC assigns a color set to each of the replacing places in RP as follows: $RC(p'_4) = RC(p'_5) = D$. The arc expression function RE assigns an expression to each of the arcs in RA as follows: $RE(t_3,p'_4) = x+2`y$, $RE(t_3,p'_5) = 2`x+y$, $RE(t_4,p'_4) = x$, $RE(p'_4,t_5) = 2`x+y$, $RE(p'_4,t_6) = y$, $RE(p'_5,t_5) = x$ and $RE(p'_5,t_6) = x+ y$.

For convenience in description, T is divided into two disjoint sets of transitions AT and UT, where AT (*affected transitions*) denotes those connected to at least one arc in RA, whereas UT (*unaffected transition*) denotes those not connected to any arcs in RA. Also, we use UP to denote P-RP. In Fig. 6, AT = {t_3, t_4, t_5, t_6} and UP = {p_1, p_2, p_3}. For a transition t in AT, the guard expression may be modified because of the change of its adjacent arcs. We use a guard function RG_U defined on AT to reflect such changes. Fig. 6 shows the matrix representation of the transformation described in Fig. 5.

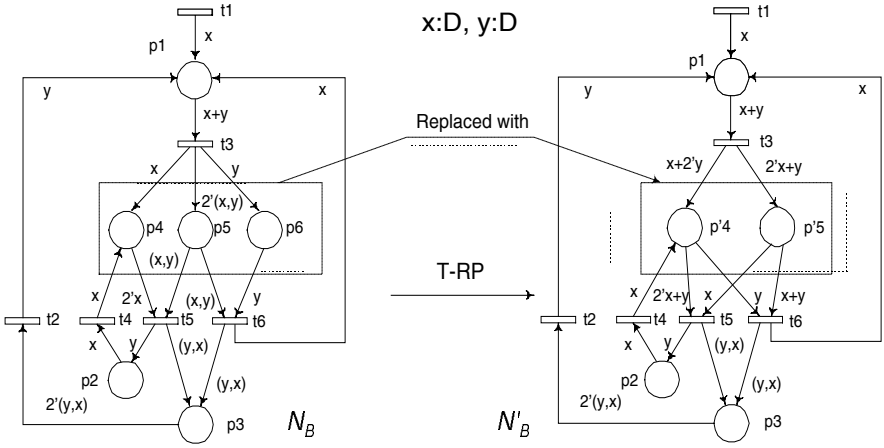


Fig. 5. T-RP transforms N_B into N'_B by replacing the set of places $AP = \{p_4, p_5, p_6\}$ with the set of places $RP = \{p'_4, p'_5\}$. Their adjacent arcs are modified as well.

		t_1	t_2	t_3	t_4	t_5	t_6
		UT		*AT*			
	p_1	x	y	$-x-y$	0	0	x
UP	p_2	0	0	0	$-x$	y	0
	p_3	0	$-2'(y,x)$	0	0	(y,x)	(y,x)
AP	p_4	0	0	x	x	$-2'x$	0
	p_5	0	0	$2'(x,y)$	0	$-(x,y)$	$-(x,y)$
	p_6	0	0	y	0	0	$-y$

		t_1	t_2	t_3	t_4	t_5	t_6
		UT		*AT*			
	p_1	x	y	$-x-y$	0	0	x
UP	p_2	0	0	0	$-x$	y	0
	p_3	0	$-2'(y,x)$	0	0	(y,x)	(y,x)
RP	p'_4	0	0	$x+2'y$	x	$-2'x-y$	$-y$
	p'_5	0	0	$2'x+y$	0	$-x$	$-x-y$

Incidence matrix of N_B Incidence matrix of N'_B

UP/UT, AP/AT: unaffected, affected places/transition; *RP:* replacing places

Fig. 6. Incidence matrix representation of the transformation in Figure 5.

Formal Description of Replacement-of-Places T-RP

Replacement-of-Places T-RP transforms a CP-net $N = <P, T, A, C, E, G>$ to another CP-net $N' = <P', T', A', C', E', G'>$ by replacing the set of places *AP* with *RP* and the set of adjacent arcs *AR* with *RA*. Associated with T-RP are a color function *RC* defined on *RP* and an arc-expression function $RE: RA \rightarrow Exps$ for assigning functions or expressions to the replacing places *RP* and arcs *RA*.

N and N' are related as follows:

a. $P' \equiv P - AP + RP$, $T' \equiv T$ and $A' \equiv A - AA + RA$,
where $AA \equiv \{A(p) \mid p \in AP\}$ and $RA \subset \bigcup_{p \in RP} (\{p\} \times T + T \times \{p\}) \cap A$

b. $C'(p) \equiv C(p)$ *if* $p \in P - AP$
$\equiv RC(p)$ *if* $p \in RP$
$\equiv RE(a)$ *if* $a \in RA$

c.. $G'(t) \equiv RG(t)$ *if* $t \in AT$, where $AT = \{ t \mid A(t) \cap AA \neq \emptyset \}$
$\equiv G(t)$ *if* $t \in UT - AT$

Replacement-of-Places T-RP can be considered as a combination of a T-EL eliminating AP and a T-IN inserting RP. However, looser conditions for preserving invariants will be given in the following subsections.

4.1 Preservation of P-Invariants under Replacement-of-Places T-RP

Theorem 3. Let N be transformed to N' by T-RP and let $U \subseteq P - AP$, $\Lambda \subseteq AP$ and $R \subseteq RP$. Suppose W_P is a place weight-function of N with support $U \cup \Lambda$ *and* W'_P is a place weight-function of N' with support $U \cup R$. If the following conditions:

a. $\forall p \in U,\ W_P(p) = W'_P(p)$ and (4.1)

b. $\forall t \in AT:\ NF(t, R, W'_P) = NF(t, \Lambda, W_P)$ (4.2)

are satisfied, then W_P is a P-invariant of N iff W'_P is a P-invariant of N'.

Proof: Let $UP = P\text{-}AP$, $UT = T\text{-}AT$, $|T| = n$ and $|UT| = i$. Then, $|AT| = n - i$. By the definition of T-RP, the incidence matrix I will be transformed to I' as shown in Fig. 7.

$$\begin{array}{c} \\ UP \\ AP \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} UT & & & AT & & \end{array} \\ \begin{pmatrix} X_1 & \dots & X_i & X_{i+1} & \dots & X_n \\ 0_1 & \dots & 0_i & Y_{i+1} & \dots & Y_n \end{pmatrix} \end{array} \longrightarrow \begin{array}{c} \\ UP \\ RP \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} UT & & & AT & & \end{array} \\ \begin{pmatrix} X_1 & \dots & X_i & X'_{i+1} & \dots & X'_n \\ 0_1 & \dots & 0_i & Y'_{i+1} & \dots & Y'_n \end{pmatrix} \end{array}$$

Incidence matrix I of N Incidence matrix I' of N'

UP/UT, AP/AT: unaffected, affected places/transition; *RP* : replacing places

Fig. 7. Representation of Replacement-of-places (T-RP) in vertical vector format.

In Fig. 7, $X_1, \dots, X_i, X_{i+1}, \dots, X_n, X'_{i+1}, \dots, X'_n$ are vertical $|UP|$-vectors and $Y_{i+1}, \dots, Y_n, Y'_{i+1}, \dots, Y'_n$ are vertical $|AP|$-vectors. Similar to those in Fig. 3, X_k and X'_k are related in such a way that their jth elements $(X_k)_j$ and $(X_k)_j$ are represented by the same expression. However, $(X_k)_j$ is defined on $B(t_j)_{MS}$ in N whereas $(X'_k)_j$ is defined on $B(t_j)_{MS}$ in N'. Note that, for $t_j \in AT$, $A(t_j)$ is changed by the transformation, causing also changes to $Var(t_j)$. Let V_j/V'_j denote $Var(t_j)$ in N/N', respectively. Suppose b and b' are two bindings of t_j in N and N', respectively. If $\forall v \in V_j \cap V_j'$ and $b(v) = b'(v)$, then

$$(X_k)_j(b) = (X'_k)_j(b'). \quad (4.3)$$

Let W_P be of the form $(\alpha\ \beta)$, where α is a $|UP|$-vector such that $\{ a \in UP \mid |\alpha(a)| \neq 0 \} = U$ and β is a $|AP|$-vector such that $\{ b \in AP \mid |\beta(b)| \neq 0 \} = \Lambda$. Then, the support of W_P is $U \cup A$. From the structural relationship of their incidence matrices, W'_P can be expressed as $(\alpha\ \beta')$, where β' is a $|RP|$-vector such that $\{ v \in RP \mid |\beta'(v)| \neq 0 \} = R$. Next, we shall prove that, if W_P and W'_P satisfy (4.2), the following formula is true:

$$W_P * I = \mathbf{0} \text{ iff } W'_P * I' = \mathbf{0}. \quad (4.4)$$

By Definition 8, we have: $\forall\ t_k \in AT$, $i < k \leq n$, $NF(t_k, A, W_P) = \beta * Y_k$, and $NF(t_k, R, W'_P) = \beta' * Y'_k$. From (4.2), we have $\beta * Y_k = \beta' * Y'_k$.

a. For $k=1,2,\dots,i$, it is obvious that $(\alpha\ \beta) * \begin{pmatrix} X_k \\ 0_k \end{pmatrix} = (\alpha\ \beta') * \begin{pmatrix} X_k \\ 0_k \end{pmatrix}$.

b. For $k = i+1, \ldots, n$, from (4.3), we have $\alpha * X_k = \mathbf{0}$ iff $\alpha * X'_k = \mathbf{0}$. Together with (3.26), we have $\alpha * X_k + \beta * Y_k = \mathbf{0}$ iff $\alpha * X'_k + \beta' * Y'_k = \mathbf{0}$.

Hence, we have $(\alpha \quad \beta) * \begin{pmatrix} X_k \\ Y_k \end{pmatrix} = 0 \text{ iff } (\alpha \quad \beta') * \begin{pmatrix} X_k \\ Y'_k \end{pmatrix} = 0$.

Hence, if the conditions described in (4.2) are satisfied, (4.3) is true. □

Corollary 3. Let *N* be transformed to *N'* by T-RP.

a. Suppose W_P is a *P*-invariant of *N* with support *INV*. Let $U = (P - AP) \cap INV$ and $\Lambda = AP \cap INV$. Suppose W'_P be a place weight-function of *N'* with support $U + R$, where $R \subseteq RP$. If *U*, Λ and *R* satisfy (4.1) and (4.2), then W'_P is a *P*-invariant of *N'*.

b. Suppose W'_P is a *P*-invariant of *N'* with support *INV*. Let $U = (P - AP) \cap INV$ and $R = RP \cap INV$. Suppose W_P is a place weight-function of *N* with support $U + \Lambda$, where $\Lambda \subseteq AP$. If *U*, Λ and *R* satisfy (4.1) and (4.2), then W_P is a *P*-invariant of *N*.

As a special case of Theorem 3, Corollary 3 makes it easier to find a *P*-invariant of the original or transformed net from each other.

4.2 Preservation of T-Invariants under Replacement-of-Places T-RP

In this section, conditions for preserving T-invariants are given in Theorem 4 whereby T-invariants of the transformed net can be found from those of the original net. But, the inverse process has not been found yet.

Theorem 4. Let *N* be transformed to *N'* by T-RP. If W_T is a *T*-invariant of *N*, then W_T is also a *T*-invariant of *N'*, provided that the following conditions are satisfied:

a. $\forall t \in T, B(t)$ is not changed under the transformation,

b. $p'_m \in RP, \ \forall t_j \in AT, \ E'(t, p_m) - E'(p_m, t) =$

$$\sum_{p_k \in AP} Z_{m,k} * \left(E(t_j, p_k) - E(p_k, t_j) \right) \tag{4.5}$$

where $Z_{m,k}$ is a function: $C(p_k)_{MS} \rightarrow C(p'_m)_{MS}$.

Proof: By Theorem 4, the incidence matrix *I* is transformed to *I'* as shown in Fig. 8.

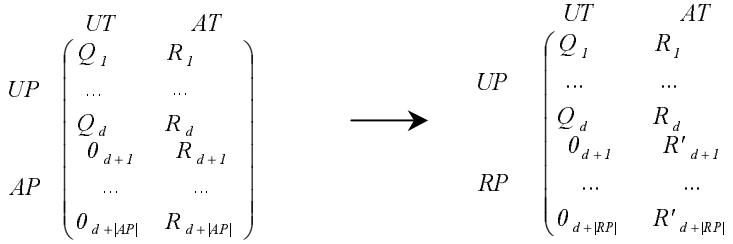


Fig. 8. Representation of Replacement-of-places T-RP in horizontal vector format.

Next, we express W in the form $(\alpha \beta)$, where α is a $|UT|$-vector and β is a $|AT|$-vector. Now we have to prove $I' * W = \mathbf{0}$. Since $I * W_T = \mathbf{0}$, we have

$$Q_k * \alpha + R_k * \beta = \mathbf{0} \quad \text{for } k = 1, \ldots, d \tag{4.6}$$

and $$R_k * \beta = \mathbf{0} \quad \text{for } k = d+1, \ldots, d+|AP|. \tag{4.7}$$

By (4.5), we have $(R'_m)_i = E'(t_{i+|UT|}, p_m) - E'(p_m, t_{i+|UT|}) =$

$$\sum_{k=d+1}^{d+|AP|} Z_{m,k} * \left(E(t_{i+|UT|}, p_k) - E(p_k, t_{i+|UT|})\right) = \sum_{k=d+1}^{d+|AP|} Z_{m,k} * (C_k)_i$$

for $m = d+1, \ldots, d+|RP|$ and $i = 1, \ldots, |AT|$.

Hence, $$R'_m * \beta = \sum_{i=1}^{|AT|} (R'_m)_i * \beta_i = \sum_{i=1}^{|AT|} \left(\sum_{k=d+1}^{d+|AP|} Z_{m,k} * (R_k)_i \right) * \beta_i =$$

$$\sum_{k=d+1}^{d+|AP|} \left(\sum_{i=1}^{|AT|} Z_{m,k} * (R_k)_i * \beta_i \right). \tag{4.8}$$

From (4.7), we have $\sum_{i=1}^{|AT|} (R_k)_i * \beta_i = \mathbf{0}$ and $\sum_{i=1}^{|AT|} Z_{m,k} * (R_k)_i * \beta_i = \mathbf{0}$. That means $\sum_{k=d+1}^{d+|AP|} \left(\sum_{i=1}^{|AT|} Z_{m,k} * (R_k)_i * \beta_i \right) = \mathbf{0}$. From (4.8), we have $R'_m * \beta = \mathbf{0}$. Together with (4.6), it follows that $I' * W = \mathbf{0}$.

Theorem 4 is useful for checking whether a T-invariant is preserved under T-RP. Take Fig. 6 as an example. Suppose $Z_{4,4}(x) = x$, $Z_{4,5}(x, y) = y$, $Z_{4,6}(y) = \mathbf{0}$, $Z_{5,4}(x) = \mathbf{0}$, $Z_{5,5}(x, y) = x$, and $Z_{5,6}(y) = y$. Then, the *T-invariant* $W_T =$

$$\begin{pmatrix} t_2\ t_3\ t_5\ t_6 & t_4 \\ <x=a,y=b>+<x=b,y=a> & <x=a>+<x=b> \end{pmatrix}$$ of the original net is preserved under T-RP.

4.3 Replacement-of-Transitions T-RT

As the dual of T-RP, T-RT transforms a CP-net to another by replacing a set of transitions and their adjacent arcs with another set of transitions and arcs. There exist results similar to Theorems 3, 4 and Corollary *3*. The details are omitted.

5 Composition T-CO and Decomposition T-DC

Composition T-CO connects two CP-nets by fusing some of their places or transitions in a one-to-one manner, while decomposition T-DC is the inverse process. They are often used in the modular analysis of a system. T-CO includes two dual classes: Composition-by-Fusing-Places T-CP and Composition-by-Fusing-Transitions T-CT.

Example. Fig. 9 shows an example of T-CP, where N_C and N_D are combined into N_E by fusing places $p_{1,1}$, $p_{2,1}$ into p'_1 and places $p_{1,2}$, $p_{2,2}$ into p'_2. As shown below, the transformation described in Fig.9 can be expressed in terms of a composition of incidence matrices (Fig. 10). Let AP_1/AP_2 (*affected places*) be the set of places in N_C/N_D to be fused, UP_1/UP_2 (*unaffected places*) be the set of places in N_C/N_D not to be fused and AA_1/AA_2 (*affected arcs*) be the arcs in N_C/N_D connected to the affected places, i.e., $AA_1 = \{ (p,t) \mid p \in AP_1\} + \{(t,p) \mid p \in AP_1\}$, $AA_2 = \{(p,t) \mid p \in AP_2\} + \{(t,p) \mid p \in AP_2\}$. A pair of affected places is fused into a new *combined place*. Let the set of combined places be AP and the set of arcs connected to the combined place be AA. In

Fig. 9, $UP_1 = \{p_1, p_2, p_3\}$, $AP_1 = \{p_{1,1}, p_{1,2}\}$, $AA_1 = \{(p_{1,1}, t_2), (t_3, p_{1,1}), (p_{1,2}, t_3), (t_4, p_{1,2})\}$, $UP_2 = \{p_4, p_5\}$, $AP_2 = \{p_{2,1}, p_{2,2}\}$, $AP = \{p'_1, p'_2\}$, $AA = \{(p'_1, t_2), (p'_1, t_6), (t_3, p'_1), (t_5, p'_1), (t_7, p'_1), (p'_2, t_3), (p'_2, t_6), (t_4, p'_2), (t_7, p'_2)\}$.

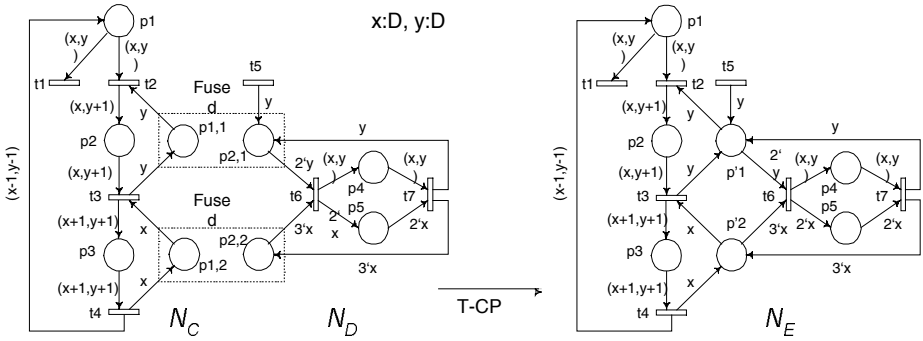


Fig.9. N_C and N_D are combined into N_E by fusing $p_{1,1}$, $p_{2,1}$ into p'_1 and $p_{1,2}$, $p_{2,2}$ into p'_2.

$$
\begin{array}{cc} & \begin{matrix} t_1 & t_2 & t_3 & t_4 \end{matrix} \\ \begin{matrix} UP_1 \\ \\ \\ \cdots\cdots \\ AP_1 \end{matrix} & \begin{matrix} p_1 \\ p_2 \\ p_3 \\ \cdots\cdots \\ p_{1,1} \\ p_{1,2} \end{matrix} \begin{pmatrix} -(x,y) & -(x,y) & 0 & (x-1,y-1) \\ 0 & (x,y+1) & -(x,y+1) & 0 \\ 0 & 0 & (x+1,y+1) & -(x+1,y+1) \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ 0 & -y & y & 0 \\ 0 & 0 & -x & x \end{pmatrix} \end{array}
$$

Incidence matrix N_C

$$
+ \quad \begin{array}{cc} & \begin{matrix} t_5 & t_6 & t_7 \end{matrix} \\ \begin{matrix} AP_2 \\ \\ \cdots\cdots \\ \\ UP_2 \end{matrix} & \begin{matrix} p_{2,1} \\ p_{2,2} \\ \cdots\cdots \\ p_4 \\ p_5 \end{matrix} \begin{pmatrix} y & -2`y & y \\ 0 & -3`x & 3`x \\ \cdots & \cdots\cdots & \cdots\cdots \\ 0 & (x,y) & -(x,y) \\ 0 & 2`x & -2`x \end{pmatrix} \end{array}
$$

Incidence matrix of N_D

$$
\rightarrow \quad \begin{array}{cc} & \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{matrix} \\ \begin{matrix} UP_1 \\ \\ \\ \cdots\cdots \\ AP \\ \\ \cdots\cdots \\ \\ UP_2 \end{matrix} & \begin{matrix} p_1 \\ p_2 \\ p_3 \\ \cdots\cdots \\ p'_1 \\ p'_2 \\ \cdots\cdots \\ p_4 \\ p_5 \end{matrix} \begin{pmatrix} -(x,y) & -(x,y) & 0 & (x-1,y-1) & 0 & 0 & 0 \\ 0 & (x,y+1) & -(x,y+1) & 0 & y & -2`y & y \\ 0 & 0 & (x+1,y+1) & -(x+1,y+1) & 0 & -3`x & 3`x \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots & \cdots\cdots \\ 0 & -y & y & 0 & y & -2`y & y \\ 0 & 0 & -x & x & 0 & -3`x & 3`x \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots & \cdots\cdots \\ 0 & 0 & 0 & 0 & 0 & (x,y) & -(x,y) \\ 0 & 0 & 0 & 0 & 0 & 2`x & -2`x \end{pmatrix} \end{array}
$$

Incidence matrix of N_E

UP_1/UP_2: unaffected places of N_C/N_D; AP_1/AP_2: affected places of N_C/N_D;
AP: places in N_E which are formed by fusing two places in AP and AP .

Fig. 10. Incidence matrix representation of the transformation in Figure 9.

Formal Description of Composition-by-Fusing-Places T-CP

Consider two CP-nets $N_1 = <P_1, T_1, A_1, C_1, E_1, G_1>$ and $N_2 = <P_2, T_2, A_2, C_2, V_2, E_2, G_2>$. Let $P_1 = UP_1 + AP_1$, where $AP_1 = \{p_{1,1}, p_{1,2}, \ldots p_{1,i}, \ldots p_{1,m}\}$ and $P_2 = UP_2 + AP_2$, where $AP_2 = \{p_{2,1}, p_{2,2}, \ldots p_{2,i}, \ldots p_{2,m}\}$. Suppose, for i = 1, 2,… , m, $p_{1,i}$ and $p_{2,i}$ have the

same set of token-colors, i.e., $C_1(p_{1,i}) = C_2(p_{2,i})$. Composition-by-Fusing-Places T-CP combines N_1 and N_2 into a CP-net N' = <P', T', A', C', E', G'> by fusing $p_{1,i}$ and $p_{2,i}$ into p'_i (i = 1, 2, ... , m).

N , N and N' are related in the following way:

a. $P' \equiv UP + AP + UP$, where $AP = \{p', p', \ldots p', \ldots p'_m\}$.

b. $T' \equiv T + T$.

c. $A' \equiv (A - AA) + (A - AA) + AA$,
where $AA = \{A(p) \mid p \in AP\}$, $AA = \{A(p) \mid p \in AP\}$ and

$$AA \equiv \sum_{i=1}^{m} \sum_{t \in T1} \left(\{(p'_i, t) \mid (p_{1,i}, t) \in A_1\} + \{(t, p'_i) \mid (t, p_{1,i}) \in A_1\} \right) +$$

$$\sum_{i=1}^{m} \sum_{t \in T2} \left(\{(p'_i, t) \mid (p_{2,i}, t) \in A_2\} + \{(t, p'_i) \mid (t, p'_{2,i}) \in A_2\} \right)$$

d. $C'(p) \equiv C(p)$ if $p \in UP$
$\equiv C(p)$ if $p \in UP$
$\equiv C(p)$ if $p = p'$, for $1 \le i \le m$.

e. $E'(a) \equiv E(a)$ if $a \in A - AA$
$\equiv E(a)$ if $a \in A - AA$
$\equiv E(p, t')$ if $a = (p', t')$, $i = 1, 2, \ldots, m$, $t' \in T$.
$\equiv E(t', p)$ if $a = (t', p')$, $i = 1, 2, \ldots, m$, $t' \in T$.
$\equiv E(p, t')$ if $a = (p', t')$, $i = 1, 2, \ldots, m$, $t' \in T$.
$\equiv E(t', p)$ if $a = (t', p')$, $i = 1, 2, \ldots, m$, $t' \in T$.

f. $G'(t) \equiv G(t)$ if $t \in T$
$\equiv G(t)$ if $t \in T$.

5.1 Preservation of Invariants under Composition-by-Fusing-Places T-CP

This section presents our results concerning the preservation of *P*-invariants and *T*-invariants under T-CP.

Theorem 5. Let T-CP combine N_1 and N_2 to form N' by fusing the places $A_1 = \{p_{1,1}, \ldots p_{1,i}, \ldots, p_{1,k}\}$ of N with the places $A_2 = \{p_{2,1}, \ldots p_{2,i}, \ldots p_{2,k}\}$ of N_2, resulting in the set of combined places $A = \{p'_1, \ldots p'_i, \ldots, p'_k\}$, where $1 \le k \le m$. Let $U_1 \subseteq UP$ and $U_2 \subseteq UP_2$.

a. Suppose W is a place weight-function of N with support $U + A$, W is a place weight-function of N with support $U + A$ and W' is a place weight-function of N' with support $U + A + U$. If the following conditions:

1. $\forall p \in U$, $W'(p) = W(p)$, (5.1)
2. $\forall p \in U$, $W'(p) = W(p)$ and (5.2)
3. for $i = 1, 2, \ldots, k$, $W(p) = W(p) = W'(p')$ (5.3)

are satisfied, then W is a *P*-invariant of N and W is a *P*-invariant of N iff W is a *P*-invariant of N'.

b. Suppose W is a transition weight-function of N with support Υ, W is a transition weight-function of N with support Υ and W' is a transition weight-function of N' with support $\Upsilon + \Upsilon$. If the following conditions:

1. $\forall t \in \Upsilon$, $W'(t) = W(t)$, (5.4)

2. $\forall t \in \Upsilon$, $W'(t) = W(t)$ and (5.5)
3. W is a T-invariant of N and W is a T-invariant of N

are satisfied, then W' is a T-invariant of N'.

Proof: As shown in Fig. 11, the incidence matrices of N_1 and N_2 are combined to form the incidence matrix of N', where $|T_1| = n_1$, $|T_2| = n_2$. Note that, for $k = 1, 2, \ldots, n_1$, $X_{1,k}$ is a vertical $|UP_1|$-vector and $Y_{1,k}$ is a vertical $|AP_1|$-vector. For $k = 1, 2, \ldots, n_2$, $Y_{2,k}$ is a vertical $|AP_2|$-vector and $X_{2,k}$ is a vertical $|UP_2|$-vector, where $|AP_1| = |AP_2| = |AP|$.

(a) Represent W as $(\alpha\ \beta)$, where α is a $|UP|$-vector such that $\{p \in UP \mid |\alpha(p)| \neq \mathbf{0}\} = U$ and β is a $|AP|$-vector such that $\{p \in AP \mid |\beta(p)| \neq \mathbf{0}\} = A$. Hence, the support of W is $U + A$. From (5.3), W can be expressed as $(\beta\ \alpha)$, where α is a $|UP|$-vector such that $\{p \in UP \mid |\alpha(p)| \neq \mathbf{0}\} = U$. Hence, the support of W is $A + U$. By (5.1) - (5.3), W'_P can be expressed as $(\alpha\ \beta\ \alpha)$. It is then easy to prove that $W * I = \mathbf{0}$ and $W * I = \mathbf{0}$ iff $W' * I' = \mathbf{0}$.

(b) From the structural relationship of N and N with N', W' can be written as $(W\ W)$. It is then easy to prove that if $I * W = \mathbf{0}$ and $I * W = \mathbf{0}$ then $I' * W' = \mathbf{0}$.

$$\begin{array}{c} \\ UP_1 \\ AP_1 \end{array}\!\!\begin{array}{c} T_1 \\ \begin{pmatrix} X_{1,1} \ldots X_{1,n1} \\ Y_{1,1} \ldots Y_{1,n1} \end{pmatrix} \end{array} + \begin{array}{c} \\ AP_2 \\ UP_2 \end{array}\!\!\begin{array}{c} T_2 \\ \begin{pmatrix} Y_{2,1} \ldots Y_{2,n2} \\ X_{2,1} \ldots X_{2,n2} \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ UP_1 \\ AP \\ UP_2 \end{array}\!\!\begin{array}{c} T_1 \quad\quad T_2 \\ \begin{pmatrix} X_{1,1} \ldots X_{1,n1} & 0_{2,1} \ldots 0_{2,n2} \\ Y_{1,1} \ldots Y_{1,n1} & Y_{2,1} \ldots Y_{2,n2} \\ 0_{1,1} \ldots 0_{1,n1} & X_{2,1} \ldots X_{2,n2} \end{pmatrix} \end{array}$$

Incidence matrix I of N Incidence matrix I of N Incidence matrix I' of N'

UP_1/UP_2: unaffected places of N_1/N_2. AP_1/AP_2 : affected places of N_1/N_2

AP: places in N' which is formed by fusing two places in AP_1 and AP_2.

Fig. 11. Incidence matrix transformation of Composition-by-Fusing-Places (T-CP).

Example. Theorem 5 makes it possible to find the invariants of a composite net from those of its components. Fig. 9 shows that $\begin{pmatrix} p_3 & p_{1,2} \\ x-1 & x \end{pmatrix}$ is a P-invariant of N_C and $\begin{pmatrix} p_{2,2} & p_4 & p_5 \\ x & x & x \end{pmatrix}$ a P-invariant of N_D. It follows from Theorem 5 that $\begin{pmatrix} p_3 & p'_2 & p_4 & p_5 \\ x-1 & x & x & x \end{pmatrix}$ is a P-invariant of N_E. Similarly, from the T-invariants $\begin{pmatrix} t_2 & t_3 & t_4 \\ <x=a,y=b> & <x=a,y=b> & <x=a,y=b> \end{pmatrix}$ of N_C and $\begin{pmatrix} t_2 & t_3 \\ <x=a,y=b> & <x=a,y=b> \end{pmatrix}$ of N_D, it follows that $\begin{pmatrix} t_2 & t_3 & t_4 & t_6 & t_7 \\ <x=a,y=b> & <x=a,y=b> & <x=a,y=b> & <x=a,y=b> & <x=a,y=b> \end{pmatrix}$ is a T-invariant of N_E.

5.2 Composition-by-Fusing-Transitions T-CT and Decomposition T-DC

Similar to T-CP, T-CT combines two CP-nets into one by fusing a set of transitions. There exist results similar to Theorem 5. Decomposition T-DC, the inverse of Composition T-CO, can be defined by reversing the roles of N_1, N_2 and N'. The conditions for preserving invariants are the same as those given in Theorem 5. Details are omitted here.

6 Concluding Remarks and Discussion on Applications

This paper provides the criteria for determining whether an invariant of a CP-net is preserved under five classes of very general transformations. They are nontrivial generalizations from basic Petri nets [8] and have greatly enhanced similar techniques limited to special transformations or to CP-nets with special structures. In this section, we describe two useful areas of application to strengthen this claim. However, owing to space limitation and the scope of this paper, our description will be sketchy and aims essentially at providing information on our relevant current and future research.

A. Application to the Synthesis of Flexible Manufacturing Systems (FMS)

In an FMS, its various parts, such as resources, loading stations, assembly subsystems, unloading stations, etc., co-operate under the control of a computer program. A major issue in handling FMSs is to synthesize the design of these programs. In the past, low-level Petri nets have been widely used [15,24,25,26]. However, because of the limitations of such Petri nets, synthesis has been restricted mainly to small-scaled systems. Recently, research and commercial applications have been extended to CP-nets. However, most of the extensions are for specification purposes and for special properties and transformations (mainly compositions). For example, Koh et al [18] showed that liveness and boundedness of a circuit are preserved if it is composed by fusing two components along two types of common directed paths. Also, Espeleta et al [13] studied deadlock prevention by proposing special methods for integrating the parts of an FMS.

The techniques reported in this paper have been used [9] in a use-case approach for modeling and analyzing FMSs. In this approach, invariants represent important properties of an FMS, such as P-invariants representing the resources and T-invariants representing cyclic processes. Our approach can be summarized as follows:

a. Specify each part of an FMS as a CP-net. If possible, express its properties under study as invariants of the CP-net. Find such invariants.
b. Modify the CP-nets by Elimination (Section 3) if functional abstraction is desired or by Insertion (Section 3) if functional refinement is desired.
c. Combine the modified CP-nets by T-CP (Section 5). Derive the invariants of the combined net from those of the modified component nets.
d. Check whether the derived invariants are the same as those for the original FMS.

B. Application to Management of Feature Interactions (FI) in Telephony Industry

A *feature*, such as *Call Waiting, Call Forwarding*, etc, is a package of services or properties for enhancing the POTS (plain old telephone service) of a telephone system. *Feature Interaction* is the phenomenon that two features, while working well individually, cause abnormality when activated simultaneously [4,5,6,21]. Managing FIs is a very complex task because they may arise in many formats, such as non-determinism, deadlock or infinite looping, loss of original functionality, etc. The causes of FIs are also very diversified, ranging from violation of feature assumptions, limitation of network supports, to the intrinsic inadequacy of distributed systems. In the telephone industry, FIs are mainly discovered through exhaustive testing and resolved through adding modifications by experts. Such an ad hoc approach becomes impractical as the number and complexity of features increases greatly under commercial competition.

Recently, formal methods have been used to avoid, detect and resolve FIs. Kawarasaki et al [17] used T-invariants of CP-nets to reduce FIs. However, since they could not find T-invariants, they had to translate a CP-net to a colorless one. Nakamura et al [22] used *P*-invariants of predicate/transition nets to check the reachability of those states where interaction occurs. The method is efficient because the scope to be detected is limited. However, finding P-invariants become an issue specially for a complex system like telephone network. Cheung et al [6,21] first proposed the approach of using property-preserving transformations in the study of FIs. This paper provides the theoretical foundation for such an approach. A summary of this approach will be given below:

a. Specify the basic system and the features each as a CP-net. As a use-case, some common behavior of the feature is represented by a set of T-invariants, and the functionality of each feature is specified as a temporal formula.

b. To integrate features into the system, we first modify the CP-net of the system by replacing a set of transitions with another set. We then insert the CP-nets of the features into the modified CP-net. Find the T-invariants of the integrated Petri net.

c. Deduce a set of firing sequences realizing the T-invariants of the integrated Petri net. A FI exists if any of the temporal formulas is violated over one of these firing sequences.

References

[1] M.L. Benalycherif and C. Girault, "Behavioural and structural composition rules preserving liveness by synchronization for colored FIFO nets", *Lecture Notes in Computer Science*, Vol. 1091, Springer-Verlag, 1980, pp.73-92.

[2] G. Berthelot, "Transformations and decompositions of nets", *Lecture Notes in Computer Science*, Vol.254, Springer-Verlag, 1987, pp.359-376.

[3] E. Best and T. Thielke, "Orthogonal transformations for colored Petri nets", *Lecture Notes in Computer Science*, Vol.1248, Springer-Verlag, 1997, pp.447-466.

[4] T.F. Bowen, F.S. Dworack, C.H. Chow, N. Griffeth, G.E. Herman, and Y.-J.Lin, "The feature interaction problem in telecommunications systems", *Proc. 7th IEE Int. Conf. on Software Engineering Telecomm. Switching Systems*, 1989, pp.59-62.

[5] E.J. Cameron, N. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Schnure and H. Velthuijsen, "A feature-interaction benchmark for IN and beyond", *IEEE Communication Magazine*, Vol.26, No.3, March, 1993, pp.64-69.

[6] T.Y. Cheung and Y. Lu, "Detecting and resolving the interaction between telephone features Terminating Call Screening and Call Forwarding by colored Petri-nets", *Proc. 1995 IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, Oct. 1995, pp.2245-2250.

[7] T.Y. Cheung, "Petri nets for protocol engineering", *Journal of Computer Communications*, Vol. 19, No. 14, Dec. 1996, pp.1250-1257.

[8] T.Y. Cheung and W. Zeng, "Invariant-preserving transformations for the verification of place/transition systems", *IEEE Trans. on System, Man and Cybernetics*, Vol. 28, No.1, Jan. 1998, pp.114-221.

[9] T.Y. Cheung and Y. Lu, "A use case based approach to sythesis and analysis of flexible manufacturing systems", Technical Report TR-98-12, Dept. of Computer Science, City University of Hong Kong, 1998.

[10] S. Christensen and L. Petrucci, "Towards a modular analysis of coloured Petri nets", *Lecture Notes in Computer Science*, Vol. 616, Springer-Verlag, 1992, pp.113-133.
[11] J. Ezpeleta and J. M. Colom, "Automatic synthesis of colored Petri nets for the control of FMS", *IEEE Trans. Robotics and Automation*, Vol. 13, No.3, June 1997, pp.327-337.
[12] S. Haddad, "A reduction theory for coloured nets", *Advances in Petri Nets 1989, Lecture Notes in Computer Science*, Vol. 424, Springer-Verlag, 1990, pp.209-235.
[13] K. Jensen, "How to find invariants for coloured Petri nets", *Lecture Notes in Computer Science,* Vol. 118, Springer-Verlag, 1981, pp.327-338.
[14] K. Jensen, "Coloured Petri nets: A high level language for system design and analysis", *Lecture Notes in Computer Science*, Vol. 483, Springer-Verlag, 1990, pp.342-416.
[15] M.D. Jeng and F. DiCesare, "A review of synthesis techniques for Petri nets with applications to automated manufacturing systems", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 23, No.1, Jan/Feb 1993, pp.301-312.
[16] *K. Jensen, Coloured Petri Net 3*, Springer-Verlag, 1997.
[17] Y. Kawarasaki and T Ohta, "A new proposal for feature interaction detection and elimination", In *Feature interactions in Telecommunication Systems III* (K. E. Cheng and T. Ohta, eds.), IOS Press, 1995, pp. 127-139.
[18] I. Koh and F. DiCesare, "Synthesis rules for colored Petri nets and their applications to automated manufacturing system", *Proc. 1991 IEEE International Symposium on Intelligent Control*, Aug. Virginia, pp.152-157.
[19] C. Lakos, "On the abstraction of coloured Petri nets", *Lecture Notes in Computer Science*, Vol. 1248, Springer-Verlag, 1997, pp.42-61.
[20] H.K.Lee, "Generalized Petri net reduction method", *IEEE Trans. System, Man, and Cybernetics*, Vol. SMC-17, No.2, March/April 1987, pp.297-302.
[21] Y. Lu and T. Y. Cheung, "Feature Interactions of the livelock type in IN: a detailed example", *Proc. 7th IEEE Intelligent Network Workshop*, Bordeaux, 1998, pp.175-184.
[22] M. Nakamura, Y. Kakuda and T. Kikuno, "Petri-net based detection method for non-deterministic feature interactions and its experimental evaluation", In *Feature Interactions in Telecommunications Systems IV* (P. Dini, R. Boutaba and L. Logrippo, eds), IOS Press. 1997, pp. 138-152.
[23] Y. Narahari and N. Viswanadham, "On the invariants of coloured Petri nets", *Lecture Notes in Computer Science*, Vol. 222, Springer-Verlag, 1986, pp.330-345.
[24] M. Silva and R. Valette, "Petri nets and flexible manufacturing", *Lecture Notes in Computer Science*. Vol. 424. Springer-Verlag, 1990, pp.374-417.
[25] M.C. Zhou, F. DiCesare, and A.A. Desrochers, "A hybrid methodology for synthesis of Petri net models for manufacturing systems", *IEEE Trans. Robotics and Automation*, Vol. 8, No.3, Jun. 1992, pp.350-361.
[26] R. Zurawski, "Verifying correctness of interfaces of design models of manufacturing systems using functional abstractions", *IEEE Trans.Industrial Electronics*, Vol.44, No.3, Jun. 1997, pp. 307-320.

Verifying Intuition – ILF Checks DAWN Proofs

Thomas Baar[1⋆], Ekkart Kindler[2⋆⋆], and Hagen Völzer[2⋆⋆⋆]

[1] Humboldt-Universität zu Berlin, Institut für Mathematik
D-10099 Berlin, Germany
[2] Humboldt-Universität zu Berlin, Institut für Informatik
D-10099 Berlin, Germany

Abstract The DAWN approach allows to model and verify distributed algorithms in an intuitive way. At a first glance, a DAWN proof may appear to be informal. In this paper, we argue that DAWN proofs are formal and can be checked for correctness fully automatically by automated theorem provers. The basic technique are proof rules which generate proof obligations. For the definition of the proof rules we adopt assertions and we introduce conflict formulas for algebraic Petri nets. Experiments show that the generated proof obligations can be automatically checked by theorem provers.

Introduction

It is a hard task to present a formal proof in an intuitively understandable way. Often, many details which are necessary for formal correctness obscure the basic idea of the proof. As a result, proofs are either formalistic rather than formal or proofs are sloppy rather than intuitively understandable.

The *Distributed Algorithms Working Notation* (DAWN) [24,21,14,22] was developed to model and verify distributed algorithms in an intuitive way. Proofs are on a high level of abstraction which helps to focus on the basic idea rather than on formalistic details (e.g. [13,22,9]). Due to the high level of abstraction, some details might be easily overlooked—resulting in an understandable but wrong proof. In this paper, we show how to avoid this problem: We introduce concepts and techniques which allow to automatically check DAWN proofs for correctness. Missing formal details (*proof obligations*) in DAWN proofs can automatically be added. With these details added, the proof can be checked by help of automated theorem provers. This way, we build a bridge from DAWN proofs to automated theorem provers and we reconcile the requirement of intuitive understandability and formality of proofs.

The basic idea for the generation of the proof obligations are proof rules based on *assertions* and *conflicts formulas*, which syntactically characterize conflicting

⋆ supported by DFG: Project 'Deduktion für Fremdnutzer' within the 'Schwerpunktprogramm Deduktion'
⋆⋆ supported by DFG: Projects 'Petri Net Technology' and 'Datenkonsistenzkriterien'
⋆⋆⋆ supported by DFG: Project 'Konsensalgorithmen'

S. Donatelli, J. Kleijn (Eds.): ICATPN'99, LNCS 1639, pp. 404–423, 1999.
© Springer-Verlag Berlin Heidelberg 1999

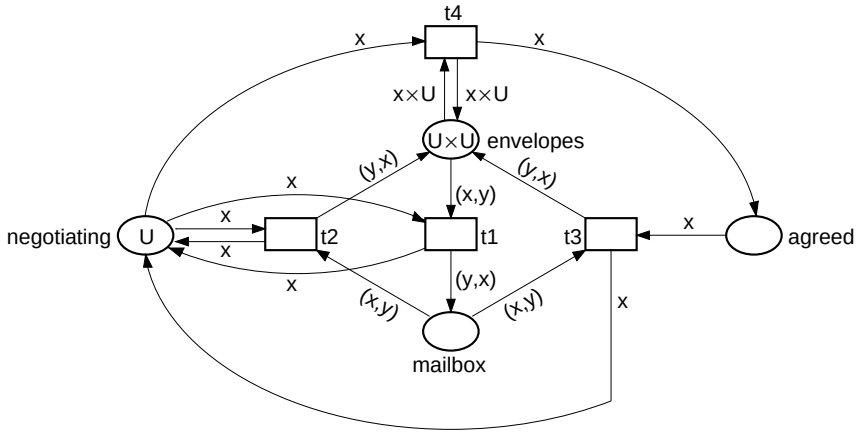


Figure1. A negotiation protocol

transitions. Our experiments have shown that theorem provers can automatically prove the generated *proof obligations* within a reasonable time. For our experiments, we have used the ILF-system[1] [8,7] which provides an interface to many automated theorem provers.

1 The Example

Before introducing the concepts in detail, we briefly introduce DAWN by an example. The example shows how to model and verify distributed algorithms in DAWN. We consider a simple protocol of negotiating agents (cf. [24,21]).

1.1 Modelling

In DAWN, distributed algorithms are modelled by a particular kind of algebraic Petri nets [12,14]. The model of the negotiation protocol is shown in Fig. 1.

We assume that there is a set of agents U participating in the negotiations. Moreover, we assume that agents do not meet personally, but communicate by sending and receiving messages. Each agent $x \in U$ may adopt two states: Either, it agrees to the current state of the negotiations or it still wants to negotiate. In the Petri net model, these two states are represented by either a token x on place agreed or a token x on place negotiating. Initially, all agents are negotiating which is represented by the inscription U at place negotiating.

The possible *actions* of a negotiating agent $x \in U$ are represented by the transitions t1, t2, and t4:

t1: An agent $x \in U$ may make a new proposal to some other agent $y \in U$. In order to send a new proposal, agent x uses a dedicated envelope. In the

[1] ILF is an acronym for Integrating Logical Functions.

system net, an envelope is modelled by a token (x, y) on place envelopes, where x is the sender and y is the receiver. Sending the envelope is modelled by removing the token (x, y) from place envelopes and putting a token (y, x) on place mailbox. Note that we do not represent the contents of a message in this model.

t2: Another action of a negotiating agent $x \in U$ is taking a message from its mailbox. Then, it also returns the envelope to the sender y of this message. As we have seen before, a message from y to x is represented by a token (x, y) on place mailbox.

t4: The last action of a negotiating agent $x \in U$ is to agree to the current result of the negotiations. But, it may only do so if all its envelopes have been returned. The arc-inscription $x \times U$ (short for $\{x\} \times U$) guarantees that every envelope of x has been returned.

An agent $x \in U$ which has already agreed to the result of the negotiations must resume the negotiations, when it receives a message from some agent. Upon receipt of the message, it returns the envelope to the sender. This action is modelled by transition t3.

Altogether, the system net from Fig. 1 models the complete behaviour of the negotiation protocol, though we have not fixed the set of agents U yet. Indeed, the protocol works for any concrete choice of U. We use this kind of *parameterization* for modelling and verifying algorithms for any number of agents and different kinds of communication networks.

1.2 Specification

Now, we show how to formalize properties of the algorithm by help of two examples. The first property is: Whenever all agents $x \in U$ have agreed, there is no message left in any agent's mailbox. This property is denoted by the temporal formula

$$\Box(\forall x \in U : \mathsf{agreed}(x)) \Rightarrow (\forall y, z \in U : \neg \mathsf{mailbox}(y, z))$$

where $\mathsf{agreed}(x)$ means that a token x is on place agreed and $\mathsf{mailbox}(y, z)$ means that a token (y, z) is on place mailbox in a given state. The other operators are interpreted as usual. The preceding temporal operator $\Box$ (read 'always') says that the formula holds true in every reachable state. Therefore, this property is called an *invariant*. The second property is: Whenever there is a message from agent y in the mailbox of agent x, agent x will eventually resume the negotiations. This is denoted by

$$x, y \in U : \mathsf{mailbox}(x, y) \rhd \mathsf{negotiating}(x)$$

The temporal operator $\rhd$ (read 'leads-to') says that on each state satisfying $\mathsf{mailbox}(x, y)$, there will eventually follow a state satisfying $\mathsf{negotiating}(x)$.

In DAWN, we have some more operators (e.g. 'unless'). In this paper, however, we concentrate on invariants and leads-to formulas. The concepts presented in this paper can be easily transferred to the other operators of DAWN.

(1) $\Box\,\mathsf{negotiating} + \mathsf{agreed} = U$	Place invariant
(2) $\Box\,\mathsf{negotiating} \times U + \mathsf{envelopes} \geq U \times U$	Individual trap
(3) $\Box\,\mathsf{envelopes} + \overline{\mathsf{mailbox}} = U \times U$	Place invariant
(4) $\Box(\forall x \in U : \mathsf{agreed}(x)) \Rightarrow (\forall y, z \in U : \neg\mathsf{mailbox}(y, z))$	Weakening (1), (2), (3)

Table1. A proof of the invariant

1.3 Verification

The above properties seem to be pretty obvious—at least after some time of thinking over the protocol. Still, these properties should be verified. Table 1 shows a DAWN proof of the first property. The proof is organized in lines. Each line consists of three parts: the line number, the proven property, and the proof argument. For example, the proof argument in lines (1) and (3) state that the invariants follow from the net's place invariants $\mathsf{negotiating} + \mathsf{agreed}$ and $\mathsf{envelopes} + \overline{\mathsf{mailbox}}$ where place invariants are represented as linear expressions as introduced in [14]: place symbols are bag-valued variables. In a given marking, these variables are assigned to the marking of the corresponding place. The expression $\overline{\mathsf{mailbox}}$ denotes the bag in which each pair of mailbox occurs reversed. The proof argument Individual trap of line (2) refers to a particular version of a trap generalized to high-level nets [14] which will be explained in more detail later. Basically, the individual trap says that in each reachable marking the expression $\mathsf{negotiating} \times U + \mathsf{envelopes}$ contains every pair $(x, y) \in U \times U$. Line (4) says, that the invariant immediately follows from the invariants proven in (1)–(3): Assume that all agents have agreed. Then, we know by (1) that no agent is negotiating any more, which implies that $\mathsf{negotiating} \times U$ is empty, too. By (2), we know $\mathsf{envelopes} \geq U \times U$. In combination with (3) follows that $\overline{\mathsf{mailbox}}$ is empty.

Though, we gave some additional arguments in the text and omitted arguments for place invariants and individual traps, we claim that the proof of Table 1 is formal and complete: We will show how to automatically add all missing details and how to check these details by help of automated theorem provers.

Before, let us briefly discuss the proof of the leads-to property which is shown in Table 2. The proof argument refers to a so-called *proof graph* which has been first introduced[2] in [20]. For a state which satisfies $\mathsf{mailbox}(x, y)$ we know by invariant (1) that either agent x is still negotiating or has already agreed. This proof argument is represented in the proof graph by the annotation Refl (1) at node 1. From a state which satisfies $\mathsf{mailbox}(x, y) \wedge \mathsf{agreed}(x)$ we know that the occurrence of transition t3 will eventually establish $\mathsf{negotiating}(x)$ which is captured by argument Progress t3 at the arc leaving node 2.

In the rest of this paper, we present techniques which allow to check the correctness of the above DAWN proof in a fully automatic way. Assertions (Sect. 3)

[2] called *proof lattice* by Owicki and Lamport

(5) $x, y \in U$: mailbox$(x, y) \triangleright$ negotiating(x) Proof graph A

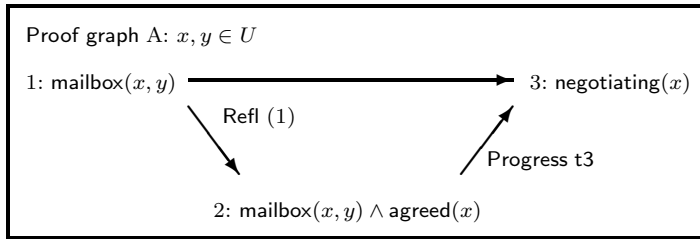


Table2. A proof of the leads-to property

and conflict formulas (Sect. 4) are used in proof rules (Sect. 5) for generating proof obligations from a DAWN proof. In Sect. 6, we will present all obligations for the above example along with experimental results of some theorem provers. In Sect. 7, we argue why the experimental results are relevant for other examples, too. Note, that we do not proof our results in this paper. The proofs can be found in the extended version of this paper [2].

2 Formalization of the Model

This section formally introduces the two central elements of DAWN: Algebraic system nets and temporal formulas. Formal definitions of prerequisites such as bags, algebras, signatures, variables, and terms can be found in the Appendix. Readers familiar with algebraic system nets and temporal logic can skip this section at first reading.

Bag-signatures and -algebras. Let $SIG = (S, OP)$ be a signature and $GS, BS \subseteq S$; $BSIG = (S, OP, bs)$ is a *bag-signature* iff $bs : GS \to BS$ is a bijective mapping. An element of GS is called *ground-sort*, an element of BS is called *bag-sort* of $BSIG$. A SIG-algebra $\mathcal{A}$ is a *$BSIG$-algebra* if for each $s \in GS$ holds $A_{bs(s)} = \mathbf{B}(A_s)$, where $\mathbf{B}(A_s)$ denotes the set of all bags over A_s.

In the following, we assume that a bag-signature $BSIG$ has a sort symbol $bool \in S$ and in each $BSIG$-algebra the corresponding domain is $A_{bool} = \{\text{true}, \text{false}\}$. Furthermore, we assume that, for each bag-sort, the usual bag operations (e.g. $\cdot + \cdot$, $[\cdot]$,$[]$) are predefined. A bag-signature $BSIG = (S, OP, bs)$ is a specialized signature $SIG = (S, OP)$ and by definition each $BSIG$-algebra is a SIG-algebra. Therefore, variables, terms, assignments, evaluations, and substitutions are defined for bag-signatures, too.

If X is a $BSIG$-variable set, then the set of all $BSIG$-terms over X of sort s is denoted by $\mathbf{T}_s^{BSIG}(X)$, the set of all terms of any sort is denoted by $\mathbf{T}^{BSIG}(X)$, and the set of all ground terms of sort s is denoted by $\mathbf{T}_s^{BSIG}$.

Algebraic system nets. Let $BSIG = (S, OP, bs)$ be a bag-signature with bag-sorts BS. An *algebraic system net* $\Sigma = (N, \mathcal{A}, X, i)$ over $BSIG$ consists of

1. a finite net $N = (P, T, F)$ where P is sorted over BS, i.e., $P = (P_s)_{s \in BS}$ is a bag-valued $BSIG$-variable set,
2. a $BSIG$-Algebra $\mathcal{A}$,
3. a sorted $BSIG$-variable set X disjoint from P,
4. a *net inscription* $i : P \cup T \cup F \to \mathbf{T}^{BSIG}(X)$ such that
   (a) for each $p \in P_s : i(p) \in \mathbf{T}_s^{BSIG}$,
   (b) for each $t \in T : i(t) \in \mathbf{T}_{bool}^{BSIG}(X)$, and
   (c) for each $t \in T$, and for each $p \in P_s$ with $f = (t, p) \in F$ or $f = (p, t) \in F$ holds $i(f) \in \mathbf{T}_s^{BSIG}(X)$.

For a place $p \in P$, the inscription $i(p)$ is called *symbolic initial marking* of p; for a transition $t \in T$ the term $i(t)$ is called *guard* of t. Note that a place is considered to be a variable and the sort of a place is a bag-sort.

Pre- and post-substitution. For each transition t of an algebraic system net Σ, we define the two substitutions $t^-, t^+ : P \to \mathbf{T}^{BSIG}(X)$, called *pre- and post-substitution* respectively, by:

$$t^-(p) = \begin{cases} i(p,t) \text{ if } (p,t) \in F \\ [] \quad \text{otherwise} \end{cases} \qquad t^+(p) = \begin{cases} i(t,p) \text{ if } (t,p) \in F \\ [] \quad \text{otherwise} \end{cases}$$

Markings, actions, and occurrence. Let $BSIG$ be a bag-signature and Σ be an algebraic system net. A *marking* M *of* Σ is an assignment for P. The marking M_0 with $M_0(p) = \beta_\emptyset(i(p))$ for each $p \in P$, where $\beta_\emptyset$ denotes the evaluation of ground terms, is called the *initial marking of* Σ. We define addition and inclusion of markings elementwise as for bags. The empty marking is denoted by θ.

Transitions of algebraic system nets occur in *modes*. A *mode* μ is an assignment for X. If μ is a mode and $t \in T$ is a transition, then $t.\mu$ is called an *action*. With each action $t.\mu$, we associate two markings μt^- and μt^+, called premarking and postmarking[3], respectively. In the following, we only consider algebraic system nets in which for each transition t and each mode μ, the markings μt^- and μt^+ are nonempty (see [4] for further explanation).

By help of these markings, we define enabledness and occurrence of actions as follows: In a given marking M_1, action $t.\mu$ is *enabled* if there exists a marking M such that $M_1 = \mu t^- + M$ and $\mu(i(t)) = \text{true}$. Then, $t.\mu$ may occur, which results in the *successor marking* $M_2 = M + \mu t^+$. We denote the occurrence of $t.\mu$ by $M_1 \xrightarrow{t.\mu} M_2$.

Remark 1. In the following, we employ terms over mixed variable sets, for example a term $\varphi \in \mathbf{T}_{bool}^{BSIG}(P \cup Y)$. Since P and Y are assumed to be disjoint, an assignment M for P (i.e. a marking) and an assignment β for Y can be canonically composed to an assignment for $P \cup Y$ which is denoted by M_β.

[3] μt^- denotes the sequent application of substitution t^- and evaluation μ.

Temporal formulas. Let Σ be an algebraic system net and Y be a variable set. A boolean term $\varphi \in \mathbf{T}_{bool}^{BSIG}(Y \cup P)$ is a *state formula*; φ is valid in a marking M under assignment β of variables Y, denoted by $M_\beta \models \varphi$ iff $M_\beta(\varphi) = \text{true}$. If φ and ψ are state formulas and $y \in Y$, then $\neg\varphi$, $\varphi \wedge \psi$, and $\forall y : \varphi$ are state formulas; validity for them is defined as usual. Note that validity of state formulas is defined with respect to some algebra $\mathcal{A}$. Let algebra $\mathcal{A}$ be fixed from now on; thus, validity means validity w.r.t. $\mathcal{A}$.

If φ and ψ are state formulas, then $\Box\,\varphi$ and $\varphi \rhd \psi$ are temporal formulas; $\Box\,\varphi$ is valid in a run $\rho = (K, r)$ under β if for all states Q of ρ we have $(r(Q))_\beta \models \varphi$, where $r(Q)$ denotes the marking associated with Q in ρ. A formula $\varphi \rhd \psi$ is valid in ρ under β if for each state Q of ρ with $(r(Q))_\beta \models \varphi$ there is a state Q' of ρ such that Q' is reachable from Q and $(r(Q'))_\beta \models \psi$. We say a temporal formula Φ is valid in Σ, denoted $\Sigma \models \Phi$, if Φ is valid in each run of Σ under each assignment β.

Substitution Lemma. In the Appendix we have defined substitutions for terms, where each variable of a term is consistently replaced by a term of the same sort. We can also apply a substitution σ to a state formula φ. However, we need to take care of bound variables. All bound variables must consistently be replaced by some fresh variables which do not occur anywhere else. A formal definition of substitution for formulas requires some technical effort and is therefore omitted. We rather state one important property of substitutions when defined in a proper way, which is known as the *Substitution Lemma*: For a formula φ, a substitution σ, and an assignment β holds $\beta \models \sigma(\varphi)$ if and only if $\beta\sigma \models \varphi$.

3 Assertions for Petri Nets

Since the work of Hoare [11], *assertions* play a central role in almost all verification techniques for sequential and parallel programs [6]. Moreover, assertions are the formal basis for most temporal logics (e.g. [16,5,15,17]). For some program statement s and two state formulas φ and ψ, the assertion $\{\varphi\}\ s\ \{\psi\}$ is valid if executing statement s from a state which satisfies φ results in a state which satisfies ψ. One reason for the success of assertions are the structural verification techniques for assertions. For example, an assertion for an assignment $\{\varphi\}\ x := e\ \{\psi\}$ is valid, if and only if the state formula $\varphi[x \leftarrow e] \Rightarrow \psi$ is valid, where $\varphi[x \leftarrow e]$ results from φ by substituting expression e for every (free) occurrence of variable x. This reduction of the validity of an assertion to the validity of a state formula is a combination of Hoare's *Axiom of Assignment* and *Rule of Consequence*.

However, there is a fundamental difference between a transition of a high-level net and an assignment statement of a parallel or sequential program: A transition can occur in different modes. The distinction of different modes will be crucial for some verification techniques based on assertions. Therefore, we need to capture assertions for an action $t.\mu$ in some way. The problem, however, is that μ is semantical in nature because it refers to the algebra and not to the signature. In

order to cope with this problem, we define assertions for *action schemes*, where an action scheme is a pair of a transition t and a finite substitution σ denoted by $t.\sigma$. The action scheme $t.\sigma$ represents the set of actions $\{t.\beta\sigma \mid \beta \text{ is an assignment}\}$. As already suggested by the same notation for actions and action schemes we will not always make a clear distinction between actions and action schemes.

Definition 1 (Assertion). *Let $\Sigma = (N, \mathcal{A}, X, i)$ be an algebraic system net, let t be a transition of Σ, let Y be some set of variables, let $\sigma : X \to \mathbf{T}^{SIG}(Y)$ be a finite substitution, and let φ and ψ be state formulas (with variables P and Y). We call $\{\varphi\}\ t.\sigma\ \{\psi\}$ an* assertion. *The assertion is valid if for each two markings M and M' of Σ and each assignment $\beta : Y \to \mathcal{A}$ with $M_\beta \models \varphi$ and $M \xrightarrow{t.\beta\sigma} M'$ we have $M'_\beta \models \psi$.*

For example, the assertion $\{\mathsf{mailbox}(x, y) \wedge \mathsf{agreed}(x)\}\ \mathsf{t3}.id\ \{\mathsf{negotiating}(x)\}$ is valid for our example from Fig. 1, where *id* denotes the identical substitution. In the rest of this section, we will reduce the validity of an assertion to the validity of a state formula. To this end, we define two substitutions ${}^{\sigma}t$ and t^{σ} for a given action scheme $t.\sigma$.

Definition 2 (Pre- and Postsplit Substitutions). *For an action scheme $t.\sigma$ of an algebraic system net, we define the substitutions ${}^{\sigma}t : P \to \mathbf{T}^{BSIG}(P \cup Y)$ and $t^{\sigma} : P \to \mathbf{T}^{BSIG}(P \cup Y)$ by ${}^{\sigma}t(p) = p + \sigma t^{-}(p)$ and $t^{\sigma}(p) = p + \sigma t^{+}(p)$ for every place p of Σ.*

Basically, ${}^{\sigma}t(p)$ splits each place (resp. its marking) into two parts: $\sigma t^{-}(p)$ represents those tokens which are consumed frpm p by action $t.\sigma$; p represents the remaining tokens at p. Similarly, t^{σ} splits the place p into those tokens $\sigma t^{+}(p)$ which are added to p by the occurrence of action $t.\sigma$ and the remaining tokens at p. Now, we can reduce the validity of an assertion to the validity of a state formula.

Proposition 1. *An assertion $\{\varphi\}\ t.\sigma\ \{\psi\}$ of an algebraic system net Σ is valid, iff the following state formula is valid in the underlying algebra $\mathcal{A}$ of Σ:*

$$\models \quad ({}^{\sigma}t(\varphi) \wedge \sigma(i(t))) \Rightarrow t^{\sigma}(\psi)$$

The proof of Prop. 1 is similar to the proof of Hoare's Axiom of Assignment, which exploits the Substitution Lemma (see [2]). As an example of an application of Prop. 1, the assertion $\{\mathsf{mailbox}(x, y) \wedge \mathsf{agreed}(x)\}\ \mathsf{t3}.id\ \{\mathsf{negotiating}(x)\}$ can be reduced to the validity of the following formula:

$$((\mathsf{mailbox} + [(x, y)])(x, y) \wedge (\mathsf{agreed} + [x])(x))) \Rightarrow (\mathsf{negotiating} + [x])(x)$$

This formula is obviously true because x occurs in the bag $(\mathsf{negotiating} + [x])$.

4 Conflict Formulas

The idea for proving a leads-to property $\varphi \rhd \psi$ is quite simple: We choose some action $t.\sigma$ which is enabled in each state satisfying φ. Moreover, this action should establish ψ when it occurs, which can be expressed by $\{true\}\ t.\sigma\ \{\psi\}$. The problem, however, is that another action $u.\tau$ could occur which disables action $t.\sigma$ and does not establish ψ. Then, we say that the two actions are *in conflict*. On the other hand, it could be that all actions in conflict to $t.\sigma$ also establish ψ. If this is proven, then we also proved $\varphi \rhd \psi$. In order to argue on those actions which are in conflict to $t.\sigma$, we introduce a *conflict formula* $\textsc{conflict}(t.\sigma, u.\tau)$ which is true in those states in which both actions may occur, but are in conflict. With this formula, we formalize that the occurrence of conflicting actions also establish ψ by: for all $t' : \{\textsc{conflict}(t.\sigma, t'.new)\}\ t'.new\ \{\psi\}$. The substitution *new* replaces each variable of the net by a fresh variable.

Before introducing conflict formulas, let us rephrase the definition of a conflict. Two actions are in conflict in a state if the premarkings of the two actions are not disjoint. Additionally, we require for two actions being in conflict that they have different effects, i.e. different premarkings or different postmarkings.

Definition 3 (Conflict). *Let Σ be an algebraic system net and M be a marking of Σ. Two actions $t.\mu$ and $u.\nu$ are* in conflict *in M if both actions (i) are enabled in M, (ii) do not have disjoint premarkings, i.e. $\mu t^- \cap \nu u^- \neq \theta$, and (iii) do not have the same effect, i.e. $\mu t^- \neq \nu u^-$ or $\mu t^+ \neq \nu u^+$.*

For an action scheme $t.\sigma$ the state formula $\textsc{enabled}(t.\sigma)$, defined by

$$\textsc{enabled}(t.\sigma) = (\bigwedge_{s \in {}^\bullet t} \sigma t^-(s) \leq s) \wedge \sigma(i(t))$$

characterizes enabledness of an action specified by $t.\sigma$. Disjointness of the corresponding premarkings is expressed by

$$\textsc{disjoint}(t.\sigma, u.\tau) = \bigwedge_{s \in {}^\bullet t \cap {}^\bullet u} \sigma t^-(s) \cap \tau u^-(s) = [].$$

The state formula $\textsc{same}(t.\sigma, u.\tau)$, defined below, expresses that two actions have the same effect:

$$\textsc{same}(t.\sigma, u.\tau) = \bigwedge_{s \in {}^\bullet t \cup {}^\bullet u \cup t^\bullet \cup u^\bullet} \sigma t^-(s) = \tau u^-(s)\ \wedge \sigma t^+(s) = \tau u^+(s)$$

Together we get

$$\textsc{conflict}(t.\sigma, u.\tau) = \begin{array}{l} \textsc{enabled}(t.\sigma)\ \wedge \textsc{enabled}(u.\tau)\ \wedge \\ \neg\textsc{disjoint}(t.\sigma, u.\tau)\ \wedge \neg\textsc{same}(t.\sigma, u.\tau) \end{array}$$

Proposition 2. *Let Σ be an algebraic system net, M be a marking of Σ, and β be an assignment. Then, we have $M_\beta \models \textsc{enabled}(t.\sigma)$ if and only if $t.\beta\sigma$ is enabled in M, and we have $M_\beta \models \textsc{conflict}(t.\sigma, u.\tau)$ if and only if $t.\beta\sigma$ and $u.\beta\tau$ are in conflict in M.*

Examples. In our example, $\mathsf{t1}.\sigma$ and $\mathsf{t3}.\tau$ are not in conflict for any substitutions σ and τ. Actions $\mathsf{t2}.\sigma$ and $\mathsf{t3}.\tau$ are in conflict in the initial marking if $\sigma(x) = \tau(x)$. A transition can be in conflict with itself as we have for $\mathsf{t2}.\sigma$ and $\mathsf{t2}.\tau$ if $\sigma(x) = \tau(x)$ and $\sigma(y) \neq \tau(y)$. Transition $\mathsf{t4}$ is never in conflict with itself.

5 Rules

In this section, we present the basic proof rules of DAWN. These rules are based on assertions and conflict formulas. For each proof argument in a DAWN proof, there is one proof rule which can be used to generate all proof obligations necessary for checking the formal correctness of the argument. This way, the proof rules formalize and generalize Reisig's *pick-up rules* [21].

Since the generation of proof obligations is going to be automated, the rules are structured in such a way that all premises of the rule can be generated from the rules name, the formula to be proven, and some additional parameters. The allowed parameters depend on the particular rule. But, there is one parameter which occurs in almost every rule: a list of already proven invariants. In an application of a rule, we refer to these invariants by a list of line numbers in which these invariants have already been proven (cf. Weakening in Table 1). In the rule itself, we refer to the conjunction of this list of invariants (where $\Box$ has been stripped) by symbol I. If no invariant is given as a parameter for I, then *true* is the default.

5.1 Rules for Invariants

Let us start with two standard rules for verifying invariant properties—Invariance and Weakening:

Invariance I : invariant (optional)
1. $\models i(\varphi)$
2. for all $t \in T$: $\{\varphi \wedge I\}\ t.new\ \{\varphi\}$

$\Box\,\varphi$

Weakening I : invariant (optional)
1. $\models I \Rightarrow \varphi$

$\Box\,\varphi$

Rule Invariance says that $\Box\,\varphi$ holds, if φ holds in the initial state (1.) and, for all possible occurrences of an action, φ holds after the occurrence of the action, if $\varphi \wedge I$ holds before the occurrence. Note that the use of substitution *new* guarantees that the rule captures every action $t.\mu$. Rule Weakening allows to conclude an invariant from already known invariants by implication.

The two rules Invariance and Weakening are *semantically complete* (cf. [6]) for proving all invariants of an algebraic system net. So, from the completeness point of view, no other rules are necessary. From the perspective of intuitive proofs, however, we need further rules which capture the concepts of place invariants, individual traps, and some further Petri net techniques. Here, we restrict ourselves to a rule for place invariants and a rule for individual traps as formalized in [14]. Other techniques proposed in [14] can be formalized in the same way.

A place invariant e is a linear expression, which does not change its value for any occurrence of an action. Linearity of e can be checked syntactically. For some linear expression $e \in \mathbf{T}^{SIG}(P \cup X)$ and some term $v \in \mathbf{T}^{SIG}(X)$ we have the following rule; the renaming of variables in e by first applying substitution *new* guarantees that variables in the arc inscriptions do not collide with variables in the expression.

Place invariant
1. $\models i(e) = v$
2. for all $t \in T$: $\models i(t) \Rightarrow t^-(new(e)) = t^+(new(e))$

$\Box\, e = v$

An individual trap is some bag-valued linear expression e such that for some set V each transition which removes some item $x \in V$ from e also adds at least one item x, again. If each $x \in V$ occurs in e in the initial marking, we know that each $x \in V$ does occur in e in all reachable markings—denoted by $\Box\, e \geq V$. More precisely, we get the following rule:

Individual trap
1. $\models i(e) \geq V$
2. for all $t \in T$: $\models (i(t) \wedge t^-(new(e))(x)) \Rightarrow t^+(new(e))(x)$

$\Box\, e \geq V$

5.2 Rules for Leads-to Properties

Next, we consider proof rules for leads-to properties. We start with the so-called progress rule [24]:

Progress t
σ : optional (default *id*)
ξ : optional (default *true*)
I : invariant (optional)
1. $\{(\varphi \vee \xi) \wedge I\}\ t.\sigma\ \{\psi\}$
2. $\models \varphi \wedge I \Rightarrow \text{ENABLED}(t, \sigma)$
3. for all $t' \in ({}^\bullet t)^\bullet$: $\{(\varphi \vee \xi) \wedge I \wedge \text{CONFLICT}(t.\sigma, t'.new)\}\ t'.new\ \{\psi\}$
4. for all $t' \in T$: $\{(\varphi \vee \xi) \wedge I\}\ t'.new\ \{\varphi \vee \xi \vee \psi\}$

$\varphi \rhd \psi$

Note that for $\xi = true$ and $I = true$ (the default) the first premise simplifies to $\{true\}\ t.\sigma\ \{\psi\}$, the third premise simplifies to $\{\text{CONFLICT}(t.\sigma, t'.new)\}\ t'.new\ \{\psi\}$, and the fourth premise is always true because *true* occurs on the right-hand side of the assertion. When generating the obligations for this default case, we will generate these simplified premises and omit the fourth premise. For this special case, the correctness of rule Progress can be easily seen: By 2., we know that $t.\sigma$ is enabled when φ holds; by 1., we know that ψ is valid once $t.\sigma$ occurs. Due to the progress assumption either $t.\sigma$ or some conflicting action will occur.

By 3., we know that all occurrences of conflicting actions also establish ψ. The optional formula ξ can be used to strengthen the precondition of the assertions; ξ is valid as long as neither $t.\sigma$ nor any conflicting transitions occur. The next rule exploits the fact that leads-to is reflexive.

$$\frac{\begin{array}{l}\mathsf{Refl}\ \ I : \text{invariant (optional)}\\ 1.\ \models (\varphi \wedge I) \Rightarrow \psi\end{array}}{\varphi \rhd \psi}$$

The above two rules in combination with the rules Transitivity and (infinite) Disjunction (cf. UNITY [5]) would give us a semantically complete proof system for interleaving leads-to[4]. Again, we introduce another rule which allows to present proofs in a more intuitive way: *proof graphs* which are similar to proof lattices [20] and proof diagrams [18].

A proof graph is a acyclic finite directed graph with exactly one node without in-going arcs and exactly one node without outgoing arcs. We call these nodes *start* and *end* of the proof graph, respectively. Each node is inscribed by a state formula, and each node except the end node is inscribed by some proof argument. Let us assume that the start node is inscribed by φ and the end node is inscribed by ψ; then the proof graph verifies $\varphi \rhd \psi$, if each proof argument associated with the (non-end) nodes verifies $\varphi_0 \rhd \psi_1 \vee \ldots \vee \psi_n$, where φ_0 is the inscription of the node and ψ_i are the inscriptions of all successor nodes in the proof graph.

For example, consider the proof graph from Table 2 again. The proof argument for node 1 is the Refl rule with invariant (1). This argument proves $x, y \in U : \mathsf{mailbox}(x, y) \rhd (\mathsf{mailbox}(x, y) \wedge \mathsf{agreed}(x)) \vee \mathsf{negotiating}(x)$. The proof argument for node 2 is the Progress rule with transition t3 as parameter. In principle, a node of a proof graph could also have a proof graph as an argument; we only need to take care that there are no cyclic dependencies.

6 The Example Revisited

With the proof rules from Sect. 5, we have finished the bridge from DAWN proofs to automated theorem provers: A proof consists of a sequence of applications of DAWN rules. Checking correct application of each DAWN rule reduces to checking the validity of state formulas and assertions. The validity of an assertion, in turn, reduces to the validity of a state formula. Altogether, checking the correctness of a DAWN proof reduces to checking the validity of a set of (first order) predicate logic formulas, which are called *proof obligations*. Checking the validity of a proof obligation is the domain of automated theorem provers—there often called *proving* or *solving* a proof obligation.

In this section, we present the proof obligations which have been generated for our example. Moreover, we answer the question whether theorem provers succeed on checking these obligations. Table 3 shows all 21 proof obligations generated for

[4] By interleaving leads-to we understand the usual leads-to operator (e. g. UNITY [5]) on sequential runs rather than on non-sequential runs as in DAWN.

$$
\begin{array}{lrcl}
(1.1) & [x] + [] & = & [x] + [] \\
(1.2) & [x] + [] & = & [x] + [] \\
(1.3) & [] + [x] & = & [x] + [] \\
(1.4) & [x] + [] & = & [] + [x] \\
(1.5) & U + [] & = & U \\
(2.1) & ([x] \times U + [(x,y)])(z) & \Rightarrow & ([x] \times U + [])(z) \\
(2.2) & ([x] \times U + [])(z) & \Rightarrow & ([x] \times U + [(y,x)])(z) \\
(2.3) & ([] \times U + [])(z) & \Rightarrow & ([x] \times U + [(y,x)])(z) \\
(2.4) & ([x] \times U + [x] \times U)(z) & \Rightarrow & ([] \times U + [x] \times U)(z) \\
(2.5) & U \times U + U \times U & \geq & U \times U \\
(3.1) & [(x,y)] + \overline{[]} & = & [] + \overline{[(y,x)]} \\
(3.2) & [] + \overline{[(x,y)]} & = & [(y,x)] + \overline{[]} \\
(3.3) & [] + \overline{[(x,y)]} & = & [(y,x)] + \overline{[]} \\
(3.4) & [x] \times U + \overline{[]} & = & [x] \times U + \overline{[]} \\
(3.5) & U \times U + \overline{[]} & = & U \times U \\
(4.1) & ((\mathsf{negotiating} + \mathsf{agreed} = U) \wedge & & \\
 & (\mathsf{negotiating} \times U + \mathsf{envelopes} \geq U \times U) \wedge & & \\
 & (\mathsf{envelopes} + \overline{\mathsf{mailbox}} = U \times U)) & \Rightarrow & ((\forall x \in U : \mathsf{agreed}(x)) \\
 & & & \Rightarrow (\forall y, z \in U : \neg \mathsf{mailbox}(y,z))) \\
(5.1) & (\mathsf{negotiating} + \mathsf{agreed} = U \wedge \mathsf{mailbox}(x,y)) & \Rightarrow & (\mathsf{negotiating}(x) \vee \\
 & & & (\mathsf{mailbox}(x,y) \wedge \mathsf{agreed}(x))) \\
(5.2.1) & \mathsf{true} & \Rightarrow & (\mathsf{negotiating} + [x])(x) \\
(5.2.2) & (\mathsf{mailbox}(x,y) \wedge \mathsf{agreed}(x) & \Rightarrow & [x] \leq \mathsf{agreed} \wedge [(x,y)] \leq \mathsf{mailbox}) \\
(5.2.3.1) & ([x] \leq (\mathsf{agreed} + [x']) & \wedge & [(x,y)] \leq (\mathsf{mailbox} + [(x',y')]) \\
 & \wedge [x'] \leq (\mathsf{agreed} + [x']) & \wedge & [(x',y')] \leq (\mathsf{mailbox} + [(x',y')]) \\
 & \wedge \neg([x] \cap [x'] = [] \wedge [(x,y)] \cap [(x',y')] = []) & & \\
 & \wedge \neg([x] = [x'] \wedge [(x,y)] = [(x',y')] & \wedge & [(y,x)] = [(y',x')] \wedge [x] = [x'])) \\
 & & \Rightarrow & (\mathsf{negotiating} + [x'])(x) \\
(5.2.3.2) & ([x] \leq (\mathsf{agreed} + [x']) & \wedge & [(x,y)] \leq (\mathsf{mailbox} + [(x',y')]) \\
 & \wedge [x'] \leq (\mathsf{negotiating} + [x']) & \wedge & [(x',y')] \leq (\mathsf{mailbox} + [(x',y')]) \\
 & \wedge \neg([(x,y)] \cap [(x',y')] = []) & & \\
 & \wedge \neg([x] = [] \wedge [] = [x'] \wedge [(x,y)] = [(x',y')] & \wedge & [(y,x)] = [(y',x')] \wedge [x] = [x'])) \\
 & & \Rightarrow & (\mathsf{negotiating} + [x'])(x)
\end{array}
$$

Table3. Proof obligations for the example

the DAWN proof from Sect. 1.3 where we have omitted preceding quantifications: Each variable v occurring freely in an obligation is actually quantified by $\forall v \in U$. Although none of these obligations is a mathematical challenge, some obligations are syntactically quite complex. Often, even simple looking obligations cannot be checked by theorem provers fully automatically. Therefore, it is not clear at all whether the proof obligations generated from a DAWN proof can be proven by a theorem prover fully automatically.

In order to answer this question, we started different theorem provers in parallel competition on each of the above obligations. The competition was carried out between the provers SETHEO, SPASS, OTTER, and PROTEIN. Table 4

Task	Best prover	Time in sec.	Task	Best prover	Time in sec.
(1.1)	SETHEO	0.02	(3.1)	—	
(1.2)	SETHEO	0.02	(3.2)	—	
(1.3)	SETHEO	0.02	(3.3)	—	
(1.4)	SETHEO	0.01	(3.4)	SETHEO	0.01
(1.5)	SETHEO	0.02	(3.5)	SPASS	17.62
			(4.1)	—	
(2.1)	—		(5.1)	SPASS	1.70
(2.2)	SETHEO	0.05	(5.2.1)	SETHEO	0.02
(2.3)	SPASS	1.93	(5.2.2)	SETHEO	0.03
(2.4)	SETHEO	0.05	(5.2.3.1)	SETHEO	12.86
(2.5)	SETHEO	0.02	(5.2.3.2)	SETHEO	2.35

Table4. Experimental results without preprocessing

shows the winning prover for each obligation along with its termination time in seconds. For each obligation, a time limit of 2 minutes has been imposed. For some obligations (e.g. (2.1)), no prover was able to complete the task within this limit. These results shows that most (including syntactically complex) formulas can easily be checked. Some obligations, however, caused difficulties. In Sect. 7.2, we deal with these difficulties and present final results.

Although OTTER and PROTEIN did not win any competition, these provers solved a lot of obligations. If an obligation is solvable by a variety of provers, it belongs to a class of good solvable problems. But, there are obligations which could only be solved by the winner of the competition (e.g. 5.2.3.1, 5.2.3.2). The proof of such tasks depends on the fact whether the 'right' prover was used.

7 Beyond the Example

In the previous sections, we have introduced concepts which allow to automatically check DAWN proofs for correctness. The experimental results have shown that this approach does work with reasonable response time for many examples. Still, there are examples which cannot be checked by the approach as it stands. Further techniques are needed to cope with the remaining examples. One issue are techniques for automated theorem provers which exploit the special structure of the generated proof obligations and the structure of theories in the realm of distributed algorithms. Here, we present first ideas which already suffice for the remaining obligations from our example. These ideas also indicate the direction of future research. Another issue are further DAWN rules—in particular rules that capture the intuition of concurrency.

7.1 ILF: Combining the Power of Theorem Provers

The proof obligations generated by DAWN proof rules are not directly presentable to theorem provers. The obligations are DAWN state formulas. Some provers

like SETHEO, however, are restricted to unsorted clauses, i. e. to formulas of the form $A_1 \wedge \cdots \wedge A_n \Rightarrow B_1 \vee \cdots \vee B_m$ where A_i and B_j do not contain any logical connector. In order to solve an obligation by a prover, a transformation from DAWN state formulas into the input formats of the different provers is necessary. Such a transformation must preserve the logical entailment properties. Our transformation is two-tiered and uses sorted first order logic as a medium layer. The first transformation step from state formulas into sorted first order formulas is staight-forward. The second step from sorted first order formulas into input formats of the provers, e.g. unsorted clauses for SETHEO, requires much more effort. The techniques for prover-specific theory compilation, however, are well investigated and implemented as a part of the ILF-system.

ILF was developed to support formal proving of mathematical theorems by exploiting the power of existing automated theorem provers. Currently, the following general purpose provers are supported by ILF: OTTER [19], SPASS [25], SETHEO [10], and PROTEIN [3]. In order to prove a task, ILF launches different provers in parallel for the same task and waits for an answer of at least one prover. If no prover can find a proof within a preset timelimit, ILF provides several possibilities to simplify the proof tasks, such that a restart of the provers has greater chances for success.

The ILF user does not need to know the specifics of the integrated provers like formats for input files, etc. She only needs to know the syntax of ILF theories, basically sorted first order logic. ILF allows to find out suitable provers for a specific class of tasks. In case a 'best' prover does not exist, the parallel working mode allows to combine the advantages of all provers.

The ILF-system is highly customizable. Its graphical user interface makes experiments with provers and theories easy. If needed, ILF-theories and proofs found by the provers can be obtained in a readable form as latex-files. A script language allows the user to simulate all user interactions and to run ILF fully automatically. This property makes it easy to incorporate ILF into other systems as a powerful deductional component [1].

7.2 Techniques to Facilitate Obligation Proving

As shown in Sect. 6, the provers could not prove all obligations automatically without special preparations. The success rate of a prover mainly depends on an adequate formulation of the theory and the obligation. The search space is increased by a large theory with a lot of irrelevant axioms as well as by an obligation which requires a long proof. There are some general techniques to tackle this problem. Here, we discuss two of them: theory selection and simplification. Both techniques require specific knowledge from the application domain and should be applied as a preprocessing, prior to the invocation of the provers. ILF supports both techniques.

Theory selection. The proof obligations are generated by DAWN proof rules. Often, all obligations belonging to one proof rule are solvable in a very similar way and need almost the same axioms. Examples are (1.1) – (1.5), (2.1) – (2.4),

and (3.1) – (3.3). The latter group of tasks could not be solved by any prover without a preprocessing (cf. Table 4). After selecting a suitable subtheory (4 axioms) from the whole theory (25 axioms), the integrated provers could solve these tasks within a time less than one second.

We guess that an appropriate subtheory for the obligations generated by some DAWN proof rule can be successfully re-used for another application of the same rule. Thus, the theory for a proof obligation is basically selected according to the DAWN proof rule which generated the proof obligation.

Simplification. Due to their automatic generation, many obligations have a high potential for simplification. An example is obligation (3.2): $\forall x, y: \ [] + \overline{[(x,y)]} = [(y,x)] + \overline{[]}$. Due to the axiom $\forall bag: \ [] + bag = bag$, the left side can be simplified to $\overline{[(x,y)]}$ and a new version $\forall x, y: \ \overline{[(x,y)]} = [(y,x)] + \overline{[]}$ can be used as an equivalent obligation. This version can even be further simplified using the axioms $\overline{[]} = []$ and $\forall bag: \ bag + [] = bag$ to $\forall x, y: \ \overline{[(x,y)]} = [(y,x)]$.

ILF automatically transforms simplification axioms into suitable rewrite rules for simplifying the proof obligations. During this process, 13 of the 21 obligations were simplified to *true*, i. e. they were completely solved by the simplification process. For these obligations, ILF does not start provers, what considerably reduces the overall time of the verification.

The search times for the ramaining obligations are shown in Table 5 (obligations simplified to true are omitted). Altogether, each of the obligations except

Task	Best prover	Time in sec.	Task	Best prover	Time in sec.
(2.1)	SETHEO	9.01	(5.1)	SPASS	3.17
(2.2)	SETHEO	0.01	(5.2.2)	SETHEO	0.02
(2.5)	SETHEO	0.02	(5.2.3.1)	SETHEO	0.08
(4.1)	—	—	(5.2.3.2)	SETHEO	0.01

Table5. Experimental results after simplification

task (4.1) could be proved automatically. Task (4.1) cannot be simplified. Only after a manual theory selection, the prover SPASS could find a proof in 115 seconds. Task (4.1) is considerably harder to check than the other tasks. This task is generated by rule Weakening which, in contrast to other rules, does not exploit any net structure. Furthermore, task (4.1) does not contain other specific regularities. Proving task (4.1) makes intensive use of the underlying theory; it exploits multiset-theory and properties of operations $+$, $\geq$, and $\times$ in a rather complex way. This gives rise to a large search space.

7.3 Further DAWN Rules

Hitherto, we introduced as many DAWN rules as needed for the example. Although there are many more rules in DAWN [24], other rules than those presented here do not introduce new difficulties in theorem proving. This holds in

particular for partial order proof methods. We illustrate this by proof rule Co-Progress for partial order leads-to formulas [24]:

$$\frac{\begin{array}{l}\textsf{Co-Progress } t \\ \qquad\qquad \sigma : \text{optional (default } id) \\ \qquad\qquad I : \text{invariant} \\ 1.\ \{\varphi\}\ t.\sigma\ \{\psi\} \\ 2.\ \varphi \Rightarrow \textsc{enabled}(t.\sigma) \\ 3.\ \text{for all } t' \in ({}^\bullet t)^\bullet\text{: } \{\textsc{conflict}(t.\sigma, t'.new)\}\ t'.new\ \{\psi\} \\ 4.\ \text{for all } p \in {}^\bullet t : I \Rightarrow p[x] \leq 1\end{array}}{\varphi \rhd \psi}$$

This rule suggests that the structure of proof obligations is the same for partial order properties. All other DAWN rules [24] generate proof obligations of the same structure. Therefore, we argue that proof obligations for further DAWN rules can be automatically checked as well.

8 Conclusion

In this paper, we have bridged the gap between DAWN proofs and automated theorem provers by proof rules which are based on assertions and conflict formulas. The proof rules allow to automatically generate proof obligations from a DAWN proof. The experiments have shown that the generated obligations can be automatically checked by today's theorem provers in reasonable time.

Up to now, we generated the proof obligations by hand but according to the fixed scheme presented in this paper. Due to the positive experience with automatically checking the proof obligations, we have started to implement a tool which automatically generates the proof obligations and passes them to Ilf [23]. With this tool, DAWN proofs will be checked in an automatic way.

DAWN is based on *algebraic system nets*. The approach presented in this paper, however, also applies to other high-level net classes such as Coloured Petri nets—provided that the underlying data types are specified by a first-order theory.

Still, there are many theoretical and practical problems to be solved. Beyond the topics mentioned in Sect. 7, let us mention two further questions:

- How can we give some advice on how to correct a proof, when the verification of an obligation fails?
- Can we lift DAWN proofs to an even higher level of abstraction such that the correctness can still be checked automatically?

These questions can only be answered, when we have finished the tool which allows us to investigate many more examples of distributed algorithms.

References

1. T. Baar, B. Fischer, and D. Fuchs. Integrating Deductional Techniques in a Software Reuse Application. In: *Journal of Universal Computer Science* 1999.

2. T. Baar, E. Kindler, H. Völzer. Verifying Intuition – ILF checks DAWN proofs. Informatik-Bericht 119, Humboldt-Universität zu Berlin, March 1999.
3. P. Baumgartner and U. Furbach. Protein: A prover with a theory extension interface. In *Proc. CADE-12*, pp. 769–773. Springer, 1994.
4. E. Best and C. Fernández. *Nonsequential Processes*, *EATCS Monographs on Theoretical Computer Science* 13. Springer-Verlag, 1988.
5. K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
6. P. Cousot. Methods and logics for proving programs. In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pp. 841–993. Elsevier, 1990.
7. B. I. Dahn and J. Denzinger. Cooperating theorem provers. In *Automated Deduction – A Basis for Applications*, Volume 2, pp. 383–416. Kluwer Academic Publishers, 1998.
8. B. I. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ILF. In *Proc. CADE-14*, pp. 55–60. Springer, 1997.
9. J. Desel and E. Kindler. Proving correctness of distributed algorithms using high-level Petri nets – a case study. In *Proc. CSD 1998*, pp. 177–186, Fukushima, Japan, Mar. 1998. IEEE Computer Society Press.
10. C. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO V3.2: Recent developments (system abstract). In *CADE-12*, pp. 778–782. Springer, 1994.
11. C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, Oct. 1969.
12. E. Kindler and W. Reisig. Verification of distributed algorithms with algebraic Petri nets. In C. Freksa, M. Jantzen, and R. Valk (eds.), *Foundations of Computer Science: Potential – Theory – Cognition*, *LNCS* 1337, pp. 261–270. Springer, 1997.
13. E. Kindler, W. Reisig, H. Völzer, and R. Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Comp.*, 9:409–424, 1997.
14. E. Kindler and H. Völzer. Flexibility in algebraic nets. In J. Desel and M. Silva (eds.), *Application and Theory of Petri Nets 1998, 19^{th} International Conference*, *LNCS* 1420, pp. 345–364. Springer-Verlag, June 1998.
15. L. Lamport. The temporal logic of actions. SRC Research Report 79, Digital Equipment Corporation, Systems Research Center, Dec. 1991.
16. Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *10^{th} Annual Symposium on Principles of Programming Languages*. ACM, Jan. 1983.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems — Specification*. Springer-Verlag, 1992.
18. Z. Manna and A. Pnueli. A temporal proof methodology for reactive systems. In M. Broy (ed.), *Program Design Calculi*, Springer, pp. 287–323, 1992.
19. W. McCune. OTTER 2.0: Recent developments (system abstract). In *Proc. CADE-10*, pp. 663–664. Springer, 1990.
20. S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Prog. Lang. Syst.*, 4(3):455–495, July 1982.
21. W. Reisig. *Elements of Distributed Algorithms — Modeling and Analysis with Petri Nets*. Springer, 1998.
22. W. Reisig, E. Kindler, T. Vesper, H. Völzer, and R. Walter. Distributed algorithms for networks of agents. In W. Reisig and G. Rozenberg (eds.), *Lectures on Petri Nets II: Applications*, *LNCS* 1492, pp. 331–385. Springer, 1998.
23. S. Unger. Automatisches Überprüfen von DAWN-Beweisen. Diploma thesis, Humboldt-Universität zu Berlin, April 1999, forthcoming.

24. M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker, E. Kindler, J. Freiheit, and J. Desel. DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 88, Humboldt-Universität zu Berlin, Dec. 1997.
25. C. Weidenbach, B. Gaede, and G. Rock. Spass & Flotter, version 0.42. In *CADE-13*, pp. 141–145. Springer, 1996.

Appendix: Definitions and Notations

Sets, families, and functions. By $\mathbb{N}$, we denote the set of natural numbers with 0. For a set A, we denote the cardinality of A by $|A|$, and we denote the set of all non-empty finite sequences over A by A^+. A *family* of sets over some *index set* I is denoted by $(A_i)_{i\in I}$. The family $(A_i)_{i\in I}$ is *pairwise disjoint*, iff for each $i, j \in I$ with $i \neq j$ holds $A_i \cap A_j = \emptyset$. If $A = (A_i)_{i\in I}$ is a family of sets, then the set $\bigcup_{i\in I} A_i$ is often also denoted by A, for convenience.
Bags. A *bag* over a fixed set A is a mapping $M : A \to \mathbb{N}$ such that the set $\{x \in A \mid M(x) \neq 0\}$ is finite. We write $M[a]$ instead of $M(a)$ for the *multiplicity* of an element a in M. We define addition $+$ and inclusion $\leq$ of bags elementwise by $(M_1 + M_2)[a] = M_1[a] + M_2[a]$ and $M_1 \leq M_2$ iff $\forall a \in A : M_1[a] \leq M_2[a]$. The cardinality of a bag M is defined by $|M| = \sum_{x\in A} M[x]$. The set of all bags over A is denoted by $\mathbf{B}(A)$. We represent a bag by enumerating its elements in square brackets: $[a_1, \ldots, a_n]$. In particular, [] denotes the empty bag.
Algebras and signatures. A *signature* $SIG = (S, OP)$ consists of a finite set S of *sort symbols* and a pairwise disjoint family $OP = (OP_a)_{a\in S^+}$ of *operation symbols*. A *SIG-algebra* $\mathcal{A} = ((A_s)_{s\in S}, (f_{op})_{op\in OP})$ consists of a family $A = (A_s)_{s\in S}$ of sets and a family $(f_{op})_{op\in OP}$ of (total) functions such that for $op \in OP_{s_1\ldots s_n s_{n+1}}$ we have $f_{op} : A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$. A set A_s of an algebra is called *domain* and a function f_{op} is called *operation* of the algebra.
Variables and terms. For a signature $SIG = (S, OP)$, we call a pairwise disjoint family $X = (X_s)_{s\in S}$ with $X \cap OP = \emptyset$ a *sorted SIG-variable set*. Each *term* is associated with a particular sort. Let $X = (X_s)_{s\in S}$ be a sorted *SIG*-variable set. The *set of SIG-terms over X of sort s* is denoted by $\mathbf{T}_s^{SIG}(X)$ and inductively defined by: (1) $x \in X_s$ implies $x \in \mathbf{T}_s^{SIG}(X)$, and (2) $u_i \in \mathbf{T}_{s_i}^{SIG}(X)$ for $i = 1, \ldots, n$ and $op \in OP_{s_1\ldots s_n s_{n+1}}$ implies $op(u_1, \ldots, u_n) \in \mathbf{T}_{s_{n+1}}^{SIG}(X)$. The set of all terms (of any sort) is denoted by $\mathbf{T}^{SIG}(X)$. A term without variables is called *ground term*. We denote the set of ground terms by $\mathbf{T}^{SIG} = \mathbf{T}^{SIG}(\emptyset)$ and the set of ground terms of sort s by $\mathbf{T}_s^{SIG} = \mathbf{T}_s^{SIG}(\emptyset)$.
Evaluation of terms. For a signature $SIG = (S, OP)$, a sorted *SIG*-variable set $X = (X_s)_{s\in S}$, and a *SIG*-algebra $\mathcal{A} = ((A_s)_{s\in S}, (f_{op})_{op\in OP})$ a mapping $\beta : X \to A$ is an *assignment for X* iff for each $s \in S$ and $x \in X_s$ holds $\beta(x) \in A_s$. We canonically extend β to a mapping $\beta : \mathbf{T}^{SIG}(X) \to A$, called *$\beta$-evaluation*, by $\beta(op(u_1, \ldots, u_n)) = f_{op}(\beta(u_1), \ldots, \beta(u_n))$ for $op(u_1, \ldots, u_n) \in \mathbf{T}^{SIG}(X)$. By $\beta_\emptyset : \emptyset \to A$ we denote the unique assignment for the empty variable set.
Substitutions. Let X and Y be *SIG*-variable sets. A mapping $\sigma : X \to \mathbf{T}^{SIG}(Y)$ is called *substitution* iff $x \in X_s$ implies $\sigma(x) \in \mathbf{T}_s^{SIG}(Y)$. Analogously to evaluations, we also extend a substitution σ to a mapping $\sigma : \mathbf{T}^{SIG}(X) \to \mathbf{T}^{SIG}(Y)$ in order to apply it to terms. In case of $Y = \emptyset$, we call σ *ground substitution*. For the composition of two substitutions σ and τ we write $\sigma\tau$ or $\sigma \circ \tau$, i.e. we have $\sigma\tau(x) = \sigma \circ \tau(x) = \sigma(\tau(x))$. Analogously, we write $\beta\sigma$ or $\beta \circ \sigma$ for the composition of an assignment β and a substitution σ.
Petri nets. A *Petri net* $N = (P, T, F)$ consists of two disjoint sets P and T and a

relation $F \subseteq (P \times T) \cup (T \times P)$. An element of P is called *place*, an element of T is called *transition*, and an element of F is called *arc* of the net. A net is finite iff both, P and T, are finite.

Occurrence nets. We start with some prerequisites. Let $N = (P, T, F)$ be a net.

1. For an element $x \in P \cup T$ of N we define the *preset* of x by ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ and the *postset* of x by $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$.
2. We define the *minimal elements of* N by ${}^\circ N = \{x \in P \cup T \mid {}^\bullet x = \emptyset\}$ and the *maximal elements of* N by $N^\circ = \{x \in P \cup T \mid x^\bullet = \emptyset\}$.
3. For $x \in P \cup T$ we define the *set of predecessors* by $\downarrow x = \{y \in P \cup T \mid (y, x) \in F^+\}$, where F^+ denotes the transitive closure of the flow relation F.

Runs are defined by help of *occurrence nets*. An *occurrence net* has two main features: The flow relation is acyclic and is not branching at places. Moreover, each element of an occurrence net has only finitely many predecessors. A net $K = (B, E, <)$ is an *occurrence net* if

1. ${}^\circ K \subseteq B$ and $K^\circ \subseteq B$,
2. ${}^\circ K$ is finite and for each $e \in E$ both, ${}^\bullet e$ and $e^\bullet$ are finite,
3. for each $b \in B$ holds $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$, and
4. for each $b \in B$ the set of predecessors $\downarrow b$ is finite and $b \notin \downarrow b$.

For the sake of clarity, we use new symbols for places and transitions of an occurrence net. Moreover, we call a place of an occurrence net *condition* and we call a transition *event*. Next we define the *states* of an occurrence net. Let $K = (B, E, <)$ be an occurrence net. For subsets of conditions $Q, Q' \subseteq B$ we define the occurrence relation $\longrightarrow$ by: $Q \longrightarrow Q'$ iff there exists an event $e \in E$ such that ${}^\bullet e \subseteq Q$ and $Q' = (Q \setminus {}^\bullet e) \cup e^\bullet$. The transitive and reflexive closure of $\longrightarrow$ is denoted by $\xrightarrow{*}$. For $Q, Q' \subseteq B$ we say Q' *is reachable from* Q, if $Q \xrightarrow{*} Q'$. A subset of conditions $Q \subseteq B$ is a *state of* K, if Q is reachable from ${}^\circ K$. The set ${}^\circ K$ is called the *initial state* of K.

Runs of algebraic system nets. In a run, each condition of the occurrence net is associated with some place of the algebraic system net along with an element of the corresponding domain. This is formalized as condition labelling. Let Σ be an algebraic system net with algebra domain A, and let $K = (B, E, <)$ be an occurrence net. A mapping $r : B \to P \times A$ is a *condition labelling of* K, iff for each $b \in B$ with $r(b) = (p, a)$ it holds that $a \in A_s \Rightarrow p \in P_{bs(s)}$.

For a given condition labelling r, each finite subset $Q \subseteq B$ can be associated with a marking. We denote this marking by $r(Q)$ and define it by $r(Q) : P \to \mathbf{B}(A)$ with $r(Q)(p)[a] = |\{b \in Q \mid r(b) = (p, a)\}|$. An occurrence net together with a condition labelling is a *run* of an algebraic system net, iff

1. $r({}^\circ K) = M_0$, where M_0 is the initial marking of Σ, and
2. for each event $e \in E$ there exists an action $t.\mu$ such that $\mu(i(t)) = \text{true}$, $r({}^\bullet e) = \mu t^-$, and $r(e^\bullet) = \mu t^+$.
3. No action is enabled in $r(K^\circ)$ in any mode.

Author Index

Allmaier S. 147

Baar T. 404
Bastide R. 66
Best E. 344
Billington J. 127
Botti O. 168

Capra L. 168
Cheung T.-Y. 384
Chrząstowski-Wachtel P. 268
Ciardo G. 6
Cortadella J. 26
Couvreur J.-M. 364

De Michelis G. 282
Devillers R. 344

Fanchon J. 304

Gaeta R. 168

Haddad S. 228
Hadjicostis C.N. 188

Juhás G. 324

Kindler E. 404
Koutny M. 344
Kreische D. 147
Krogh B. H. 106
Kummer O. 86

Lavagno L. 208
Le D.-H. 66
Lu Y. 384

Miner A. S. 6
Moldt D. 86

Navarre D. 66

Palanque P. 66
Pastor E. 26
Peña M.A. 26
Poitrenaud D. 228, 364

Recalde L. 107

Sangiovanni-Vincentelli A. 208
Schmidt K. 46
Schneider C. 248
Sgroi M. 208
Silva M. 107
Sy O. 66

Teruel E. 107
Tokmakoff A. 127

Varaiya P. 1
Verghese G.C. 188
Vogler W. 284
Völzer H. 404

Watanabe Y. 208
Wehler J. 248
Wienberg F. 86