Preface

The idea of the UFO workshop emerged from the following observation: Since the proposal of the complete unfolding prefix concept by McMillan, a number of research teams have extended and applied this appealing idea in different directions. But there have been few opportunities to make a synthetic point on the flourishing results obtained so far, although some of their protagonists certainly met at conferences like ATPN, ACSD, CAV, CONCUR, FOSSACS, TACAS, etc. The maturity of the topic and the surprising variety of current research trends suggested that today might be the right time to attempt such an event.

The objective of the UFO workshop is twofold: First, we wanted to bring together researchers working on different aspects of the unfolding theory, and second, we tried to popularise unfoldings in the research and industry communities. To achieve these objectives, a large part of this workshop is devoted to a tutorial and invited talks. The remaining time slots are reserved to original contributions on the subject, focusing on research aspects, applications or tool developments.

Of course, it would not be possible to cover all current research directions related to unfoldings at a single workshop. Nevertheless, the large number of invited talks gave space to many of them, while the call for papers brought some good surprises!

Browsing the programme reveals several "main" research axes. One is definitely related to unfolding new classes of models, like high-level nets (Koutny), timed nets (Jard), or nets with read arcs (Baldan et al.), also called contextual nets. A second trend consists in defining different types of unfoldings, taking the form of symbolic unfoldings (Jard), merged processes (Khomenko), or trellis processes (Fabre). And, of course, an unavoidable concern remains the construction of finite and complete prefixes in all these cases (Baldan et al., Madalinski, Vogler).

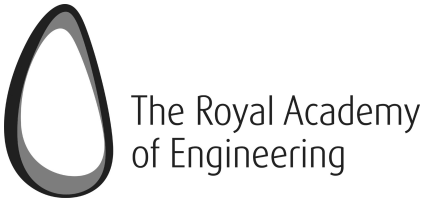
A sign of good health of the research related to unfoldings is that it also ranges from fundamental aspects (Vogler, Winskel) to various practical applications such as model checking (Esparza, Khomenko, Koutny) and synthesis of asynchronous circuits (Yakovlev), which are in fact the historical targets pondered already by McMillan. But new and unexpected applications are also appearing, such as distributed diagnosis (Fabre), or the use of unfoldings to solve planning problems (Bonet et al.), which is one of the nice surprises of this workshop.

Unfortunately, some active topics are not represented. Among these great absentees, let us mention for example the relations between unfoldings and event structures, where many interesting models have been proposed. The unfolding of more exotic models of concurrent systems like graph grammars have not been covered either. And we also regret the absence of papers about the randomisation of event structures and unfoldings, which is a very exciting and promising topic. Nevertheless, we do hope that the variety of the menu will satisfy your appetite and will stimulate your imagination.

We cannot close this introduction without warmly thanking the research leaders in this field, who kindly accepted the invitation to participate in this workshop and prepared invited presentations. The Program Committee has also appreciated the quality of some submissions. Unfortunately, space limitations imposed a selection, and we encourage the authors of rejected papers to pursue their works and bring them to maturity.

Finally, this event wouldn't have been possible without the support of the Royal Academy of Engineering (UK) and of INRIA Rennes (France), to whom we are very grateful.

Eric Fabre and Victor Khomenko, June 2007
The UFO'07 organisers.



Workshop Organisation

Programme Chairs

Eric Fabre
Victor Khomenko

Programme Committee

Jean-Michel Couvreur
Javier Esparza
Eric Fabre
Keijo Heljanko
Victor Khomenko
Claude Jard
Maciej Koutny
Christian Stehno
Walter Vogler
Alex Yakovlev

External Reviewers

Frédéric Herbreteau
Denis Poitrenaud
Grégoire Sutre

Local Organisation

Wojciech Nabialek
Artur Niewiadomski
Wojciech Penczek

Table of Contents

Canonical Prefixes of Petri Net Unfoldings

Walter Vogler[1]

Institut für Informatik, Universität Augsburg
D-86135 Augsburg, Germany
Walter.Vogler@informatik.uni-augsburg.de

1 Introduction

The unfolding *Unf* of a Petri net N gives a partial-order based representation of all behaviours and reachable markings of the net. In his seminal paper [8], McMillan presented his method to compute a complete prefix of this unfolding as a way to alleviate the problem of state space explosion. This method was improved in [2]; the ERV-algorithm introduced there generalises the one of McMillan since it is parametric in a so-called adequate order, denoted $\lhd$. These notes sketch the results and further improvements for this approach from [6], obtained by Victor Khomenko, Maciej Koutny and the present author.

In general, the ERV-algorithm is non-deterministic: at each step, among the possible extensions, some $\lhd$-minimal event e is chosen and added to the branching process π constructed so far; then, the event is marked as a cut-off, if there is an event e' in π such that $[e']$ and $[e]$ reach the same marking and $[e'] \lhd [e]$. Now it could be that such an e' exists in *Unf*, but has not been added to π, and thus e is not recognized as a cut-off in this run of the algorithm; so it seems that the notion of a "cut-off event" depends on the choices which of the possible extensions to add, i.e. it is a dynamic notion. It also seems that the ERV-algorithm could lead to different results.

In these notes, it is shown that this is in fact not the case: we define a static, i.e. algorithm-independent notion of a cut-off event (but depending on the adequate order); this leads to a unique *canonical* prefix, and we proof that this prefix is complete. We further show that the ERV-algorithm computes this prefix in each of its runs, i.e. the algorithm is non-deterministic but determinate – and this result holds more generally also for a parallel unfolding algorithm proposed in [4], called *slicing algorithm*. This gives an alternative correctness proof for both algorithms; for the slicing algorithm, this new proof is much simpler than the original one, which compared the runs of this algorithm with those of the ERV-algorithm.

The new results are obtained for a slightly stronger version of completeness than usual and also in a more general, parametric setting of *cutting contexts*. In the standard setting, one is essentially interested in the marking of a configuration; but when modelling circuits, one might be interested in additional information [9], namely the so-called state vector, and thus *fewer* configurations carry the same information; or one might only be interested in markings up to some symmetries in order to reduce the size of the prefix (see e.g. [1]), and thus

more configurations carry the same information. In the standard setting, the notion "cut-off event" only depends on local configurations; again to reduce the size of the prefix, one might also consider *all* configurations [3]. Cutting contexts allow to treat all such variations in *one* setting.

It is assumed that the reader has some knowledge of [8, 2] and the ERV-algorithm, and in particular knows the following notions: (Petri) net; branching process and its prefixes and (finite) configurations; marking of (i.e. reached by) a configuration; (finite) suffix E' of a configuration C, where this is indicated by the notation $C \oplus E'$ for their union; local configuration $[e]$, with $\langle e \rangle$ as notation for $[e] \setminus \{e\}$.

For these notes, we fix a finite net $N = (S, T, W, M_0)$ and its unfolding $Unf = (B, E, G, h)$ (which is the greatest branching process). $\mathcal{C}$ denotes the (finite) configurations of Unf, $\mathcal{C}_{loc}$ its local configurations, and $<$ the causality relation on E. For a branching process π, $\mathcal{C}^\pi$ denotes the configurations of π.

2 Cutting contexts and canonical prefixes

As explained in the introduction, a cutting context captures various possibilities for the relevant information and for the configurations used to define cut-off events. Formally, it is a tuple of three parameters.

The first parameter will determine the information we intend to preserve in a complete prefix. To capture the abstract idea of information carried by a configuration, we let the first parameter be an equivalence relation $\approx$ on $\mathcal{C}$: equivalent configurations are considered as having the same information. In the standard case, this information is the marking reached by the configuration. Thus, in the standard case, configurations with the same marking are equivalent ($\approx_{mar}$). In the other two applications mentioned above, equivalent configurations must additionally give rise to the same state vector, must only have symmetric markings resp.

The idea behind $\approx$ implies that we have to speak of equivalent configurations where one speaks of markings in the standard setting. For a complete prefix, it is sufficient to have one configuration from each equivalence class; thus, in the standard setting, every reachable marking is represented.

The third parameter specifies, separately for each event $e \in E$, which configurations can make e a cut-off; in the standard case, we have $\mathcal{C}_e = \mathcal{C}_{loc}$ for all $e \in E$. The second parameter is an adequate order and discussed below.

Definition 1. A *cutting context* is a triple $\Theta \stackrel{\text{df}}{=} \left(\approx, \lhd, \{\mathcal{C}_e\}_{e \in E}\right)$, where:

1. $\approx$ is an equivalence relation on $\mathcal{C}$.
2. $\lhd$, called an *adequate* order, is a well-founded strict partial order on $\mathcal{C}$, refining $\subset$, i.e. $C' \subset C''$ implies $C' \lhd C''$.
3. $\approx$ and $\lhd$ are *preserved by finite extensions*, i.e. for every pair of configurations $(C', C'') \in \approx$, and for every suffix E' of C', there exists a finite suffix E'' of C'' such that

   (a) $C'' \oplus E'' \approx C' \oplus E'$, and
   (b) if $C'' \lhd C'$ then $C'' \oplus E'' \lhd C' \oplus E'$.
4. $\{\mathcal{C}_e\}_{e \in E}$ is a family of subsets of $\mathcal{C}$. $\diamond$

Preservation of finite extensions is a case where the original definition [2, Definition 4.5] of adequate orders considers two configurations with the same marking, whereas we consider equivalent configurations. In [2, Definition 4.5], it is specified for 3(b) that $E'' = I_1^2(E')$, where I_1^2 is the 'natural' isomorphism between the finite extensions of C' and C''. That isomorphism is defined only if $Mark(C') = Mark(C'')$, and thus cannot be used in our generalised settings. Here, we additionally have to require 3(a), which is automatically true for $\approx_{mar}$ in the standard setting. Note that all adequate orders in the standard setting are also adequate in ours with $\approx_{mar}$ and $\mathcal{C}_{loc}$ as first and third parameter; since we allow any E'' with the same change on the marking as E', our definition is more general.

We will say that a cutting context Θ is *dense*) if $\mathcal{C}_e \supseteq \mathcal{C}_{loc}$ for all $e \in E$, and this usually is the case in practice. In the rest of these notes, we assume that a cutting context Θ is fixed.

We will write $e \lhd f$ whenever $[e] \lhd [f]$. Clearly, $\lhd$ is a well-founded partial order on the set of events. Hence, we can use Noetherian induction for definitions and proofs, i.e. it suffices to define or prove something for an event under the assumption that it has already been defined or proven for all its predecessors.

A simple idea is to define an algorithm-independent (or static) notion of a cut-off event as an event e for which there is a (so-called *corresponding*) configuration $C \in \mathcal{C}_e$ such that $C \approx [e]$ and $C \lhd [e]$. But this notion is useless in practice, since it does not go together with the idea of constructing a complete prefix: in the standard setting, it may indeed happen that a corresponding local configuration C of a cut-off event e defined in this way contains a cut-off event itself; since construction of the prefix stops at cut-offs, C cannot be in the prefix. Though in this case *Unf* contains another corresponding configuration $C' \lhd C$ with no cut-off events and the same final marking, it might be that such a configuration is never local. Thus, the ERV-algorithm will certainly not recognise e as a cut-off, and it seems that no efficient algorithm for constructing a complete prefix will.

Hence, our definition is slightly more complicated. It also defines feasible events, which are precisely those events without cut-offs in their local configurations, and as such must be included in the prefix determined by the cut-off events.

Definition 2. The set of *feasible* events, denoted by $fsble_\Theta$, and the set of *static cut-off* events, denoted by cut_Θ, are two sets of events of *Unf* defined inductively, in the following way:

1. An event e is a feasible event if $\langle e \rangle \cap cut_\Theta = \emptyset$.
2. An event e is a static cut-off event if it is feasible, and there is a configuration $C \in \mathcal{C}_e$ such that $C \subseteq fsble_\Theta \setminus cut_\Theta$, $C \approx [e]$ and $C \lhd [e]$. In what follows, any C satisfying these conditions will be called a *corresponding* configuration of e.

The branching process Unf^Θ induced by the events from $fsble_\Theta$ is called the *canonical* prefix of Unf. ◇

It is not difficult to see that for any $f \in \langle e \rangle$ or $f \in C \lhd [e]$, we have $f \lhd e$. Thus, we can assume that for the events in $\langle e \rangle$, it has already been decided whether they are in $fsble_\Theta$ or in cut_Θ, and the same holds for the events in any configuration C satisfying $C \lhd [e]$. Therefore, $fsble_\Theta$ and cut_Θ are well-defined sets by Noetherian induction.

The definition of Unf^Θ is based on the fact that $fsble_\Theta$ is downward closed w.r.t. $<$; this fact is not difficult to prove.

3 Completeness, finiteness, and algorithmics

To make canonical prefixes useful, we have to show that they are complete and, under reasonable conditions, finite. Furthermore, there must be a reasonable algorithm to compute them; in fact, we show that the ERV- and the slicing-algorithm compute exactly the canonical prefix in each run. As already remarked, this gives us alternative correctness proofs for the standard setting, but it also proves correctness for the much more general cutting contexts.

We now introduce a slightly stronger notion of completeness for branching processes than usual.

Definition 3. A branching process π is *complete w.r.t. a set* E_{cut} of events of Unf if the following hold:

1. If $C \in \mathcal{C}$, then there is $C' \in \mathcal{C}^\pi$ such that $C' \cap E_{cut} = \emptyset$ and $C \approx C'$.
2. If $C \in \mathcal{C}^\pi$ is such that $C \cap E_{cut} = \emptyset$, and e is an event such that $C \oplus \{e\} \in \mathcal{C}$, then $C \oplus \{e\} \in \mathcal{C}^\pi$.

A branching process π is *complete* if it is complete w.r.t. some set E_{cut}. ◇

Note that, in general, π remains complete after removing all events e for which $\langle e \rangle \cap E_{cut} \neq \emptyset$; i.e. without affecting the completeness, one can truncate a complete prefix so that the events from E_{cut} will be either maximal events of the prefix or not in the prefix at all. Note also that this definition depends only on the equivalence $\approx$, and not on the other components of the cutting context.

The first condition requires that each equivalence class of $\approx$ is represented by a configuration without cut-offs; in the standard setting, this exactly means that each reachable marking is represented. The second condition requires that for each configuration without cut-offs also all possible firings are represented; this is a bit stronger than the usual requirement that each reachable marking, (i.e. each equivalence class), is represented by *some* configuration without cut-offs and with all possible firings. This weaker notion is not enough to ensure correctness of the deadlock detection algorithm presented in [8]. However, the proof of completeness in [2] is actually very close to the proof of the following theorem, although our result is algorithm-independent.

Theorem 4 (completeness). *Unf^Θ is complete w.r.t. $E_{cut} = cut_\Theta$.*

The next crucial issue is finiteness. As a proof tool, we developed a version of König's Lemma (see [7]), which states that a finitely branching, rooted, directed acyclic graph with infinitely many nodes reachable from the root has an infinite path. This lemma cannot be applied to a branching process directly, since its conditions can have infinitely many outgoing arcs.

Proposition 5. *A branching process π is infinite iff it contains an infinite $<$-chain of events.*

This result implies immediately that Unf^Θ is finite iff there is no infinite $<$-chain of feasible events. This in turn is useful for proving the following result, which provides quite a tight and practical indication for deciding whether Unf^Θ is finite or not.

Theorem 6 (finiteness II).

1. *If $\approx$ has finitely many equivalence classes and Θ is dense, then Unf^Θ is finite.*
2. *If $\approx$ has infinitely many equivalence classes, then Unf^Θ is infinite.*

In particular, for the standard setting this means that Unf^Θ is finite iff N is bounded (guaranteeing finitely many reachable markings and, thus, finitely many equivalence classes).

Finally, a rather technical proof shows that the unfolding algorithm presented in [2] and the parallel slicing algorithm proposed in [4] generate the canonical prefix defined above; they are therefore correct. An essential lemma states that, before the algorithms add an event e to the prefix constructed so far, each feasible $f \lhd e$ is already in this prefix.

We come to a close with a pointer to a recent application of cutting contexts. In the context of circuit design with Petri nets, V. Khomenko studies net transformations like splitting a transition into a sequence of two transitions [5]. From the definition of usual adequate orders, it follows that such a transformation gives a very different looking complete prefix. But switching the cutting context allows to apply the same transformation to the complete prefix of the original net to obtain the complete prefix of the transformed net.

References

1. J.-M. Couvreur, S. Grivet, and Denis Poitrenaud: Unfolding of Products of Symmetrical Petri Nets. Proc. of *ICATPN'2001*, J.-M. Colom and M. Koutny (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2075 (2001) 121–143.
2. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20 (2002) 285–310.
3. K. Heljanko: Minimizing Finite Complete Prefixes. Proc. of *CS&P'99*, Workshop Concurrency, Specification and Programming (1999) 83–95.

4. K. Heljanko, V. Khomenko and M. Koutny: Parallelization of the Petri Net Unfolding Algorithm. Proc. of *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2002)*, J. -P. Katoen and P. Stevens (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2280 (2002) 371–385.
5. V. Khomenko: Behaviour-Preserving Transition Insertions in Unfolding Prefixes. Proc. of *ATPN 2007*, . Springer-Verlag, Lecture Notes in Computer Science (2007) to appear.
6. V. Khomenko, M. Koutny, and V. Vogler: Canonical Prefixes of Petri Net Unfoldings. Proc. of *International Conference on Computer Aided Verification (CAV'2002)*, E. Brinksma and K. G. Larsen (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2404 (2002) 582–595. Full version: Acta Informatica 40(2) (2003) 95-118.
7. D. König: Über eine Schlußweise aus dem Endlichen ins Unendliche. *Acta Litt. ac. sci. Szeged* 3 (1927) 121–130. Bibliography in: Theorie der endlichen und unendlichen Graphen. Teubner, Leipzig (1936, reprinted 1986)
8. K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *4th CAV*, G. von Bochmann and D. K. Probst (Eds.). Springer-Verlag, Lecture Notes in Computer Science 663 (1992) 164–174.
9. A. Semenov: *Verification and Synthesis of Asynchronous Control Circuits Using Petri Net Unfolding*. PhD Thesis, University of Newcastle upon Tyne (1997).

Modular Processings based on Unfoldings

Eric Fabre

Irisa/Inria
Campus de Beaulieu
35042 Rennes cedex, France
Eric.Fabre@irisa.fr

Extended abstract

Unfoldings, like other structures encoding the runs of concurrent systems, enjoy a nice *factorization property*. When a (labeled) Petri net can be expressed as a product of smaller components, like $\mathcal{N} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_n$, its unfolding $\mathcal{U}(\mathcal{N})$ also factorizes in a similar manner :

$$\mathcal{U}(\mathcal{N}) = \mathcal{U}(\mathcal{N}_1) \times^O \ldots \times^O \mathcal{U}(\mathcal{N}_n) \tag{1}$$

where the superscript in $\times^O$ denotes a product specific to occurrence nets. This result has been mentioned only in few publications [2,5], and remained unused for a long time. The objective of this presentation is to show how one can take advantage of it to perform modular computations on compound systems.

The first thing to notice is that factorized forms, like the right hand side in (1), are generally more compact than their extended form, *i.e.* the left hand side. The reason is that the product of labeled nets takes the disjoint union of places, but *combines* transitions that have common labels : a transition t_1 of $\mathcal{N}_1$ carrying the same label as t_2 in $\mathcal{N}_2$ will produce a synchronous change (t_1, t_2) in $\mathcal{N}_1 \times \mathcal{N}_2$. Since several transitions of a net can have the same label, the product thus multiplies transitions, as its name indicates. This holds also for the product of occurrence nets in (1). Therefore it may be more efficient to work directly on a factorized form, for applications like model-checking, reachability analysis and others.

In this talk, we propose a methodology to do so. The approach relies on an analogy between distributed systems, *i.e.* systems that can be expressed as the combination of components, and Bayesian networks, also called Markov fields or graphical models. The latter are well known statistical models that describe the interactions of large sets of random variables. Their interest is to represent graphically the structure of variable interactions, from which statistical inference algorithms can be designed. In distributed systems, interactions are governed by shared labels between components $\mathcal{N}_i$, in Bayesian networks, they are governed by local statistical coupling. Without developing this analogy, the idea is that one can find a counterpart to the notion of statistical independence in the field of distributed systems. This is sufficient to import modular inference algorithms, that perform computations at the scale of components and operate by message passing between neighboring components.

Formally, the approach relies on three ingredients. The first one is a composition operator on the objects of interest, unfoldings in our case. The product mentioned above is a natural candidate (although it's more convenient to restate it as a pullback). The second ingredient is a projection operator with respect to a (set of) component(s). For example the projection of the unfolding $\mathcal{U}(\mathcal{N})$ on its component $\mathcal{N}_1$. Again, the natural candidate is the projection that automatically comes with a product, and it works in simple cases, but a stronger notion is necessary in general. The last ingredient is a small set of axioms, that composition and projection have to satisfy. As soon as these conditions are satisfied, one can derive message passing algorithms that operate on the factorized form.

The first typical application of this setting is to compute the projections on some components $\mathcal{N}_i$ of the global unfolding $\mathcal{U}(\mathcal{N})$, starting from the unfoldings $\mathcal{U}(\mathcal{N}_1), \ldots, \mathcal{U}(\mathcal{N}_n)$, but *without computing* $\mathcal{U}(\mathcal{N})$ itself, which can be huge. This yields a generally strict prefixes of the local unfolding $\mathcal{U}(\mathcal{N}_i)$, that describes the remaining behaviors of component $\mathcal{N}_i$ once it is inserted in the global net. Another straightforward application is related to distributed diagnosis : given observations collected on different components of $\mathcal{N}$, find the behaviors of $\mathcal{N}$ that could be valid explanations to these distributed observations. Beyond its interest to reduce the complexity of computations, a very nice feature of this approach is that it generalizes to different formalisms. For example one can replace unfoldings with trellis processes [7], a close cousin of the merged processes [1] proposed by Khomenko *et al.* (also presented in this workshop).

We conclude this presentation with a related topic : the computation of finite and complete prefixes in factorized form, which corresponds to the postdoc research topic of Agnes Madalinski (see the paper "Modular construction of finite and complete prefixes" in these proceedings).

References

1. V. Khomenko, A. Kondratyev, M. Koutny, W. Vogler, Merged Processes: a New Condensed Representation of Petri Net Behaviour, Acta Informatica, Volume 43(5), pp. 307-330, Springer 2006.
2. G. Winskel, Categories of models for concurrency, Seminar on Concurrency, Carnegie-Mellon Univ. (July 1984), LNCS 197, pp. 246-267, 1985.
3. G. Winskel : Petri Nets, Algebras, Morphisms, and Compositionality , Information and Computation, vol. 72, pp. 197-238, 1987.
4. A. Benveniste, E. Fabre, S. Haar, C. Jard, Diagnosis of asynchronous discrete event systems, a net unfolding approach, IEEE Trans. on Automatic Control, vol. 48, no. 5, pp. 714-727, May 2003.
5. E. Fabre, A, Benveniste, S. Haar, C. Jard : Distributed Monitoring of Concurrent and Asynchronous Systems, Journal of Discrete Event Systems, special issue, pp. 33-84, May 2005.
6. E. Fabre : Distributed Diagnosis based on Trellis Processes, 44th Conf. on Decision and Control (CDC), Seville, Spain, 12-15 Dec. 2005.
7. E. Fabre, Trellis processes : a compact representation for runs of concurrent systems, J. of Discrete Events Dynamic Systems, March 2007.

Merged Processes of Petri Nets (invited talk)

Victor Khomenko*

School of Computing Science, Newcastle University, NE1 7RU, U.K.
E-mail: Victor.Khomenko@ncl.ac.uk

The main drawback of model checking is that it suffers from the *state space explosion* problem [8]. That is, even a relatively small system specification can (and often does) yield a very large state space. There are several common sources of state space explosion. One of them is concurrency, and the unfolding techniques (see, e.g., [1, 2, 5]) were primarily designed for efficient verification of highly concurrent systems. Indeed, complete unfolding prefixes are often exponentially smaller than the corresponding reachability graphs, because they represent concurrency directly rather than by multidimensional 'diamonds' as it is done in reachability graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the reachability graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will be isomorphic to the net itself. However, *unfoldings do not cope well with some other important sources of state space explosion, in particular with sequences of choices and non-safeness.* The examples below illustrate this problem.

First, consider Figure 1(a). The smallest complete prefix is exponential in the size of the Petri net (note that no event can be declared cut-off — intuitively, each reachable marking of the Petri net 'remembers' its past). Thus Petri nets performing a sequence of choices leading to different markings may yield exponential prefixes.

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

The described problems with Petri net unfoldings should be viewed in the light of the fact that all the above examples have a very simple structure — viz. they are all acyclic, and thus many model checking techniques, in particular those based on the *marking equation* [2, 6, 7], could be applied *directly to the original Petri nets.* And so it may happen that a prefix exponential in the size of the Petri net is built for a relatively simple problem!

* Victor Khomenko is a Royal Academy of Engineering/EPSRC Post-Doctoral Research Fellow. His research is supported by Royal Academy of Engineering/EPSRC grant EP/C53400X/1 (DAVAC).

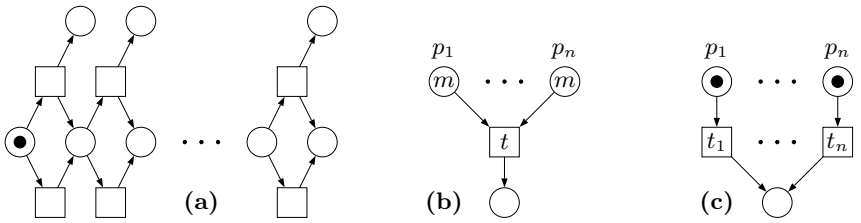


Fig. 1. Examples of Petri nets.

This talk is devoted to a different condensed representation of Petri nets' behaviour, called *merged processes,* which remedies the problems described above. It copes well not only with concurrency, but also with the other mentioned sources of state space explosion, viz. sequence of choices and non-safeness. A merged process can be obtained from a branching process by the following procedure.

1. Fuse equally labelled condition with the same *occurrence-depth*, defined for each p-labelled condition c as the maximum number of instances of p on a causal chain leading from the (virtual) initial event to c. The initial marking of the resulting *mp-condition* is set to the number of initial conditions fused onto it.
2. Delete, one-by-one, the duplicate events, i.e., ones whose label, preset and postset are equal to those of some other event (the remaining events are called *mp-events*).

For example, the unfoldings of the Petri nets shown in Figure 1(a) are collapsed back to the original nets by this procedure.

It turns out that merged processes are sufficiently similar to the traditional unfoldings, and so a large body of results developed for unfoldings can be reused. In particular, one can lift the results related to *canonicity*, *finiteness* and *marking-completeness* (see [2, 4]), as well as the upper bounds on the *size* (see [1, 2, 4]) to merged processes. Moreover, one can prove some new upper bounds on the size which hold for merged processes but not for unfolding prefixes. The conducted experiments showed that merged processes are often by orders of magnitude more compact than the corresponding unfolding prefixes, and are in most cases not much bigger than the original Petri nets.

Unfortunately, the unfolding-based model checking techniques are not directly applicable to merged processes. The main problem is that the latter can contain cycles, and the marking equation alone is no longer sufficient for characterising the reachable markings of a merged process. However, one can fix this problem for merged processes of safe Petri nets by using additional constraints besides the marked equation. This yields a general model checking approach for checking reachability-like properties. It turns out that checking most such properties is NP-complete in the size of the merged process, i.e., no worse than for unfolding prefixes. Moreover, the experimental results showed that in many cases

unfolding-based and merged processes-based model checking times are comparable.

To summarise, the proposed representation of Petri nets' behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking. This talk is based on the joint work with Alex Kondratyev, Maciej Koutny and Walter Vogler [3].

References

1. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20 (2002) 285–310.
2. V. Khomenko: *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD Thesis, School of Computing Science, Newcastle University (2003).
3. V. Khomenko, A. Kondratyev, M. Koutny and V. Vogler: Merged Processes — a New Condensed Representation of Petri Net Behaviour. *Acta Informatica* 43(5) (2006) 307–330.
4. V. Khomenko, M. Koutny and V. Vogler: Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica* 40 (2003) 95–118.
5. K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *CAV'1992*, LNCS 663 (1992) 164–174.
6. S. Melzer and S. Römer: Deadlock Checking Using Net Unfoldings. Proc. of *CAV'97*, LNCS 1254 (1997) 352–363.
7. T. Murata: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77 (1989) 541–580.
8. A. Valmari: The State Explosion Problem. In: *Lectures on Petri Nets I: Basic Models*, LNCS 1491 (1998) 429–528.

Use of Partial Orders for Analysis and Synthesis of Asynchronous Circuits

Alex Yakovlev

Asynchronous Systems Group,
School of EECE, University of Newcastle upon Tyne
Alex.Yakovlev@ncl.ac.uk
http://async.org.uk

Introduction

As semiconductor technology faces problems with designing complex ICs in the nanometre scale (65nm and beyond) asynchronous circuits are becoming increasingly incorporated in future systems on chip. Asynchronous circuits enable variability aware design and optimization of power-performance tradeoff for the high end CMOS processes. They are now actively explored in many R&D areas, such as communication and synchronization schemes in networks-on-chip (NoCs), interfaces in globally asynchronous locally synchronous (GALS), low power processors to name but a few.

One of the difficult aspects of designing and testing systems with asynchronous components is the complexity of their dynamic behaviour, particularly due to concurrency that is inherent in such systems. High degree of concurrency leads to state space explosion, the effect which hinders (a) analysis and verification of the circuits' dynamic behaviour where the properties of interest require explicit state enumeration, and (b) synthesis of the logic implementation of the circuits from their behavioural specifications, where logic functions are derived from the explicit state space covers. State-based techniques are certainly algorithmically easier to construct and implement, and as result they often lead quicker to working software tools. For example, in the analysis and verification domain, people often reduce the model-checking problem to using one of the existing state space analysers, cf. [1, 2]. In the synthesis domain, the state-based methods can easily construct next state functions and derive Boolean equations for the gates, cf. [3]. However these methods and tools cannot cope with large systems, for example one of the popular synthesis tools Petrify is limited by 30-40 binary signals, which makes such tools applicable to designing only small controllers, such for example as pipeline stage controllers. Another difficult aspect of the state-based approach is that it does not lead to easy visualization of the system's behaviour. The designer in practice is not interested in seeing the global state of the system yet being limited only by a small window in time. Instead the designer would prefer to see partial, or local, views of the system (individual

threads) but see them in their entirety. This is what happens, for instance, when resolving the complete state coding (CSC) problem. In state-based visualisation, it is not realistic to deal with systems with more than few hundred states, or about 10 signals.

This presentation will focus on the methods of analysis and synthesis of asynchronous circuits avoiding explicit state space exploration.

Partial order approach and role of Petri net unfoldings

Explicit state based models, such as state graphs or labelled transition systems, offer simpler and easier route to automated analysis and synthesis of asynchronous circuits.

In spite of this, from the early days of asynchronous circuit theory by Muller and Bartky [4] (which gives rise to the main behavioural correctness criteria for the analysis and synthesis methods, i.e. speed-independence and semi-modularity) people have been interested in exploring an alternative, or at least complementary, route which goes through event-based representations. In theory of concurrency such techniques fall into the category of partial order or true concurrency semantics. In particular, in the Petri net framework, these methods are often associated with the construction and use of a Petri net unfolding and its finite prefix. In the context of analysis of circuit specifications and logic synthesis they have been applied to Signal Transition Graphs or STGs (which are signal-event interpretation of Petri nets [5,6]).

Here is a brief (and by no means complete!) list of important developments in the application of event-based models to analysis and synthesis of asynchronous circuits:

60s:

- Flow chart and change chart methods by Gillies [7], Swartwout [8] and Shelly [9].

70s:

- Modelling of control structures using signal graphs (based on marked graphs) and their direct analysis by Jump and Thiagarajan [10].

80s:

- Circuit synthesis based on cyclo- and taxograms by Starodoubtsev [11].
- Theory, methods and tools (eg. Tranal, Traspec) for analysis and synthesis based on Change Diagrams and their unfoldings by Kishinevsky, Kondratyev, Taubin and Varshavsky [12].
- Relation-based approach to analysis of STG models of circuits by Rosenblum and Yakovlev [13].

90s:

- Petri net unfolding prefix by McMillan [14] and its application to asynchronous circuit verification.

- Analysis of STGs and circuits by Kondratyev et al [15] and Semenov [16] using unfoldings, including new ideas about cutoffs in the STG context, notion of signal deadlocks etc.
- Unfolding-based analysis of timed circuits by Semenov and Yakovlev [17]
- Synthesis from STG unfoldings by Semenov et al. [18] using unfolding cut, slice and cover approximation.
- Circuit analysis based on unfoldings for PNs with read arcs by Vogler et al. [19].

2000s:

- Analysis of STG unfoldings using LP and SAT by Khomenko et al. [20].
- Synthesis from STGs based on unfoldings and SAT solvers by Khomenko et al. [21].
- Visualization of STG-based synthesis using unfoldings by Madalinski et al. [22].
- Combining decomposition and unfolding of STGs for synthesis by Khomenko and Shaefer [23].

Following these developments with further research, there is a real chance of creating powerful tools for verification and synthesis of large scale asynchronous circuits (with 100s and 1000s of signals) and integration of these tools with the industrial strength design environments, such as Haste and Balsa.

This presentation will highlight the following main aspects of the use of Petri net unfoldings in the context of analysis and synthesis of asynchronous circuits:

- Petri net models of asynchronous circuits using level-based and event-based approaches.
- Properties requiring verification based on the reduction of hazard properties to dynamic conflicts or non-safeness.
- Problems with unfolding circuit models using Petri nets with read-arcs and k-safe nets.
- Complete State Coding analysis and visualization using unfoldings.
- Deriving logic from unfoldings.

References

[1] D. Borrione, M. Boubekeur, L.Mounier, M. Renaudin, A. Sirianni, “Validation of asynchronous circuit specifications using IF/CADP”, Proc. VLSI-SOC’03, 2003.

[2] X. Wang, M. Kwiatkowska, “On process-algebraic verification of asynchronous circuits”, Proc. ACSD’06, 2006.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Logic Synthesis of Asynchronous Controllers and Interfaces, Springer Series in Advanced Microelectronics, vol. 8, Springer, 2002, ISBN-3-540-43152-7.

[4] David E. Muller and W. S. Bartky, "A theory of asynchronous circuits", Proceedings of an International Symposium on the Theory of Switching, pages 204-243. Harvard University Press, April 1959.

[5] T.-A. Chu, C. K. C. Leung, and T. S. Wanuga, "A design methodology for concurrent VLSI systems", Proc. International Conf. Computer Design (ICCD), pages 407-410. IEEE Computer Society Press, 1985.

[6] L. Y. Rosenblum and A. V. Yakovlev. "Signal graphs: from self-timed to timed ones", Proceedings of International Workshop on Timed Petri Nets, pp.199-207, Torino, Italy, July 1985. IEEE Computer Society Press.

[7] D.B. Gillies, "A flow chart notation for the description of a speed independent control", Proc. of the 2nd Annual AIEE Symp. On Switching Circuit Theory and Logical Design", Detroit, Vol. S-134, Oct. 1961.

[8] R.E. Swartwout, "One method for designing speed-independent logic for a control", Proc. of the 2nd Annual AIEE Symp. On Switching Circuit Theory and Logical Design", Detroit, Vol. S-134, pp. 94-106, Oct. 1961.

[9] J.H. Shelly, "The decision and synthesis problems in semi-modular switching theory", Report no.88, University of Illinois, Digital Computer Lab., May 20, 1959.

[10] J.R. Jump, P.S. Thiagarajan, "On the interconnection of asynchronous control structures", J. ACM vol. 22, No. 4, pp. 596-612, 1975.

[11] N.A. Starodoubtsev, "Asynchronous processes and antitonic control circuits", Soviet Journal of Computer and Systems Science (USA) vol. 23. No. 2, pp. 112-119, No.6, pp. 81-87, Vol. 24, No.2, pp. 44-51.

[12] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky, Concurrent Hardware. The Theory and Practice of Self-Timed Design, 1994, John Wiley and Sons Ltd., 368 pp.

[13] L.Ya. Rosenblum and A.V. Yakovlev, "Analysing semantics of concurrent hardware specifications", Proc. Int. Conf. on Parallel Processing (ICPP89), Pennstate University Press, University Park, PA, July 1989, pp. 211-218, Vol.3.

[14] K.L.McMillan, "Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits". CAV 1992, pp.164-177.

[15] A. Kondratyev, M. Kishinevsky, A. Taubin, and S. Ten, "Analysis of Petri nets by ordering relations in reduced unfoldings", Formal Methods in System Design, Vol. 12, No.1, pp. 5-38, 1998.

[16] A. Semenov, Verification and Synthesis of Asynchronous Control Circuits using Petri nets Unfoldings, PhD Thesis, Department of Computing Science, University of Newcastle upon Tyne, 1998.

[17] A. Semenov and A.Yakovlev, "Verification of asynchronous circuits using time Petri-net unfolding", Proc. ACM/IEEE Design Automation Conference, pages 59-63, 1996.

[18] A. Semenov, A. Yakovlev, E. Pastor, M. Pena, J. Cortadella, and L. Lavagno, "Partial order approach to synthesis of speed-independent circuits", Proc. ASYNC'97, pp. 254-265, April 1997.

[19] W. Vogler, A. Semenov and A. Yakovlev, "Unfolding and Finite Prefix for Nets with Read Arcs", Proceedings of CONCUR'98, Nice, France, Sept. 1998, LNCS No. 1466, pp. 501-516.

[20] V. Khomenko, M. Koutny and A. Yakovlev, "Detecting State Coding Conflicts in STG Unfoldings Using SAT", Special Issue on Best Papers from ACSD'2003, IOS Press, Fundamenta Informaticae 62(2) (2004) 1-21.

[21] V. Khomenko, M. Koutny and A. Yakovlev, “Logic Synthesis for Asynchronous Circuits Based on Petri Net Unfoldings and Incremental SAT”, Special Issue on Best Papers from ACSD'2004, IOS Press, Fundamenta Informaticae 70(1-2) (2006) 49-73.

[22] A. Madalinski, A. Bystrov, V. Khomenko and A. Yakovlev, “Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design”, Special Issue on Best Papers from DATE'2003, IEE Proceedings: Computers & Digital Techniques 150(5) (2003) 285-293.

[23] V. Khomenko and M. Schaefer, “Combining Decomposition and Unfolding for STG Synthesis”, Proc. of ATPN'2007, Kleijn, J. and Yakovlev, A. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 4546 (2007) 223-243, to appear.

Branching Processes of High-Level Petri Nets and Model Checking of Mobile Systems

Maciej Koutny

School of Computing Science, Newcastle University
Newcastle upon Tyne NE1 7RU, U.K.
Maciej.Koutny@ncl.ac.uk

Abstract. This talk consists of two parts, respectively based on the papers, [4] and [5]. We first discuss branching processes and unfoldings of high-level Petri nets and outline an algorithm which builds finite and complete prefixes of such unfoldings. Its advantage is that it avoids a potentially expensive translation of a high-level Petri net into a low-level one. In the second part we present an application of such an approach to the verification of mobile systems.
Keywords: verification, model checking, high-level Petri nets, unfoldings, π-calculus, mobility.

1 Summary

Model checking suffers from the state space explosion problem, as a relatively small system specification can yield a very large state space. To cope with this, a number of techniques have been proposed, which can roughly be classified as aiming at a compact representation of the full state space of a reactive system, or at an explicit generation of its reduced (though sufficient for a given verification task) representation. Our approach is based on partial order semantics of concurrency and the corresponding Petri net unfoldings [3, 7]. A finite and complete unfolding prefix of a Petri net is a finite acyclic net which implicitly represents all the reachable states of the original net together with transitions enabled at those states. Complete prefixes are often exponentially smaller than the corresponding state graphs, especially for highly concurrent systems, because they represent concurrency directly rather than by multidimensional "diamonds" as it is done in state graphs.

In this talk, we first describe an approach introduced in [4] which allows one to build the prefix directly from a high-level Petri net [1, 6] — a compact representation of a concurrent or distributed system. Such a method is often superior to one involving the explicit construction of an intermediate low-level net as it is often the case that the intermediate low-level net is much larger than the resulting prefix. We describe branching processes and unfoldings of high-level Petri nets and an algorithm which builds finite and complete prefixes. An important relation between the branching processes of a high-level net and those of its low-level expansion is that the sets of their branching processes are the

same, allowing one to import results proven for low-level nets. The approach is conservative in the sense that all the verification tools employing the traditional unfoldings can be reused with such prefixes. Experimental results indicate that it is, on one hand, better than the traditional approach on data-intensive application, and, on the other hand, has comparable performance on control-intensive ones.

In the second part, we turn our attention to mobility which has now become a central feature of many real life concurrent and distributed computing systems. To model it and reason about its properties, process algebras such as π-calculus [8] have been introduced and studied. We describe how the model-checking technique based on Petri net unfoldings could be used in the verification of π-calculus terms. Our starting point is a compositional translation from a finite fragment of the π-calculus into a class of high-level Petri nets proposed in [2]. We developed prototype tool based on this 'theoretical' translation. It should be stressed that developing the prototype was not a straightforward task. In particular, places used for managing coloured tokens representing π-calculus channels have infinite markings and so are not directly implementable. Another problem concerned an efficient implementation of read arcs used by the theoretical translation. Experimental results suggest that model-checking based on Petri net unfoldings can be a successful technique to verify distributed systems with mobility.

References

1. E.Best, H.Fleischhack, W.Fraczak, R.Hopkins, H.Klaudel, and E.Pelz: A Class of Composable High Level Petri Nets. Proc. of ICATPN'1995, Springer, LNCS 935 (1995) 103–120
2. R.Devillers, H.Klaudel and M.Koutny: Petri Net Semantics of the Finite π-calculus Terms. Fundamenta Informaticae **70** (2006) 203–226
3. J.Engelfriet: Branching processes of Petri Nets. Acta Informatica **28** (1991) 575–591
4. V.Khomenko and M.Koutny: Branching Processes of High-Level Petri Nets. Proc. of TACAS'03, Springer, LNCS 2619 (2003) 458–472
5. V.Khomenko, M.Koutny and A.Niaouris: Applying Petri Net Unfoldings for Verification of Mobile Systems,. Report CS-TR: 953, Newcastle University (2006)
6. K.Jensen: Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use. EATCS Monographs on Theoretical Computer Science (1992).
7. K.L.McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of CAV'92, LNCS 663 (1992) 164–174
8. R.Milner, J.Parrow and D.Walker: A Calculus of Mobile Processes. Information and Computation **100** (1992) 1–77

Concurrent Operational Semantics of Safe Time Petri Nets*

Claude Jard

European University of Brittany, ENS Cachan Bretagne, IRISA
Campus de Ker-Lann, F-35170 Bruz cedex, France
Claude.Jard@bretagne.ens-cachan.fr

Abstract The paper addresses the problem of the definition and construction of finite prefixes of unfoldings of Safe Time Petri Nets. This required to consider symbolic unfoldings consisting of a graphical part (coding causal and concurrent relations) and a set of time constraints (coding the set of possible dates of firing of the transitions). The difficult problems come from nets with urgency and confusion, which reveals a certain contradiction between the global semantics of time and the idea of having a partial order semantics to avoid the usual state explosion in case of large concurrency. The paper assembles the definitions and new results we have obtained on the subject.

1 Introduction and Related Work

Time Petri Nets are one of the most popular timed extensions of Petri nets. They were first introduced in 1976 by Merlin and Farber in [9], and have proved their interest in modeling real-time concurrent systems. In Time Petri Nets, a time interval is assigned to each transition in order to restrict the possible delays between the date of its enabling and the date of its firing. Time Petri Nets can be unbounded (in the number of tokens), but boundedness is undecidable. To stay in the decidable world, and to find finite representations of the set of possible behaviours, we decide to restrict the exposition to bounded nets, and even to simplify, to safe (i.e. 1-bounded) nets. We also require that bounds on the delays are nonnegative rational numbers (denoted $\mathbb{Q}$).

Very little work has been done about unfoldings of timed true concurrency models. The reason that makes it difficult is that time yields a kind a synchronization even between parts of the net that are not connected. This clashes with a partial order semantics based on the fully asynchronous nature of the model.

The first approach in the literature was to propose a transformation from Time Petri Net to low-level ordinary Petri nets ([2] and [8]). Restricted to discrete time, the progress of time is modeled by firing special transitions called ticks.

* Most of the material presented here has been developed in the PhD document of Thomas Chatain [3] under my supervision. This work is being continuing with the help of the ANR national research program DOTS under the reference ANR-06-SETI-003

The major drawback is that all the components of the net participate in the ticks. Most of the concurrency is lost and we are faced to the classical state explosion problem. In [7], the approach has been extended to high-level nets but time progresses similarly. In [11], the notion of timed process was introduced, but only for Timed Petri Nets, which are not able to represent urgency. Time unfoldings were introduced in [10] in a particular subcase of time independent choice nets, in which urgency and confusion never appear together. We propose a solution for the general problem (nevertheless limited to safe nets).

The main originality of our approach is in defining a concurrent operational semantics for Safe Time Petri Nets, where it will be possible to fire a transition without looking at the global state of the net. Only a few tokens will give enough information to be sure that the transition can fire. Using our concurrent semantics, we can define a symbolic unfolding and a finite representation of it by a symbolic prefix. This work was partly published in [4] and [5].

This paper assembles all the definitions and results we have obtained on the subject.

2 Time Petri Nets Sequential Semantics

2.1 Time Petri Nets: Syntax

Definition 1. (graph of a simple net) *A (marked and safe)* Petri net *is a 5-tuple* $\langle P, T, pre, post, M_0 \rangle$ *where* P *and* T *are finite sets of* places *and* transitions *respectively,* pre *and* $post$ *map each transition* $t \in T$ *to its* preset *often denoted* ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \subseteq P$, $({}^\bullet t \neq \emptyset)$ *and its* postset *often denoted* $t^\bullet \stackrel{\text{def}}{=} post(t) \subseteq P$; $M_0 \subseteq P$ *is the* initial marking*, that is the set of places hosting one token at the initial step. To simplify, we assume that the preset of each transition is not empty.*

Definition 2. (graph of a time net) *A (marked and safe)* Time Petri Net *is a 7-tuple* $\langle P, T, pre, post, efd, lfd, M_0 \rangle$ *where* $\langle P, T, pre, post, M_0 \rangle$ *is a safe Petri net, and* $efd : T \longrightarrow \mathbb{Q}$ *and* $lfd : T \longrightarrow \mathbb{Q} \cup \{\infty\}$ *associate the* earliest firing delay $efd(t)$ *and* latest firing delay $lfd(t)$ *with each transition* t*.*

A Time Petri Net is represented as a graph with two types of nodes: places (circles) and transitions (boxes). The closed interval $[efd(t), lfd(t)]$ (or the half-open interval $[efd(t), \infty[$ when $lfd(t) = \infty$) is written near each transition (see Figure 1).

2.2 Time Petri Nets: Sequential Semantics

Definition 3. (global state) *A* (global) state *of a Time Petri Net is given by a triple* (M, dob, θ)*, where* $M \subseteq P$ *is a marking, and* $dob : M \longrightarrow \mathbb{Q}$ *associates a date with each marked place. This date is the latest* date of birth *of a token in the place.* θ *is the current date of the state. The net starts in its* initial marking $(M_0, dob_0, 0)$ *where* $dob_0(p) \stackrel{\text{def}}{=} 0$ *for all* $p \in M_0$*.*

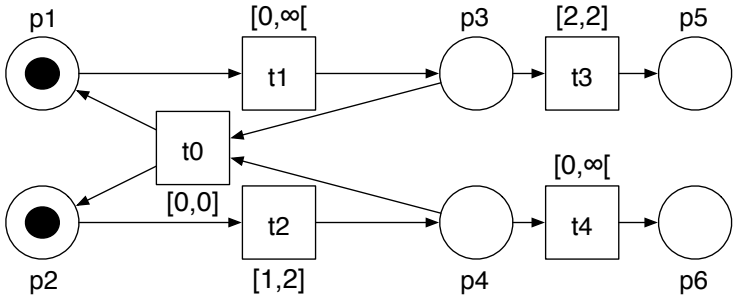


Figure 1. A Time Petri Net. A possible firing sequence is $t_2.t_1.t_0.t_1.t_2.t_3$, where possible dates of firing are $1.3, 3, 3, 3, 5, 5$ respectively. The associated symbolic constraints are $[1 \leq \theta_1] \wedge [2 \geq \theta_1])$, $[0 \leq \theta_2])$, $[\theta_3 \geq max(\theta_2, \theta_1)] \wedge [\theta_3 \leq max(\theta_2, \theta_1)] \wedge [\theta_3 \leq \theta_2 + 2]))$, $[\theta_3 \leq \theta_4] \wedge [\theta_4 \geq \theta_3 + 2])$, $[\theta_5 \geq \theta_3 + 1] \wedge [\theta_5 \leq \theta_3 + 2] \wedge [\theta_5 \leq \theta_4 + 2]))$, $[\theta_6 \geq \theta_4 + 2] \wedge [\theta_6 \leq \theta_4 + 2] \wedge [\theta_6 \leq max(\theta_4, \theta_5)]$. We can see that θ_6 depends on θ_4 (local condition), but also on θ_5, variable associated to a transition not connected to t_3, revealing the non-local aspect of time.

The *date of enabling* $doe(t)$ of a transition t is the date of birth of the youngest token in its input places (i.e. the last expected token): $doe(t) \stackrel{\text{def}}{=} max_{p \in {}^\bullet t} dob(p)$.

Definition 4. (firing rule) *A transition $t \in T$ is* enabled *in the state (M, dob, θ) and can fire at a date denoted $\theta' \geq \theta$ if:*

- *all of its input places are marked: ${}^\bullet t \subseteq M$,*
- *the minimum delay is reached: $[\theta' \geq doe(t) + efd(t)]$,*
- *all the inputs places remain marked until θ', that is the maximum delays of all enabled transitions are not overtaken:*
  $\forall t' \in T, {}^\bullet t' \subseteq M \Rightarrow [\theta' \leq doe(t') + lfd(t')]$.

The firing of t leads to the state (M', dob', θ'), where $M' = (M \setminus {}^\bullet t) \cup t^\bullet$ and $dob'(p) \stackrel{\text{def}}{=} dob(p)$ if $p \in M \setminus {}^\bullet t$ and $dob'(p) \stackrel{\text{def}}{=} \theta$ if $p \in {}^\bullet t$.

The enabling condition is denoted

$$Enabled(M, dob, t, \theta) = \begin{cases} {}^\bullet t \subseteq M \wedge [\theta \geq max_{p \in {}^\bullet t} dob(p) + efd(t)] \wedge \\ \bigwedge_{t' \in T, {}^\bullet t' \subseteq M} [\theta \leq max_{p \in {}^\bullet t'} dob(p) + lfd(t')] \end{cases}$$

The fact that a transition t has been fired is denoted $(M, dob, \theta) \xrightarrow{t} (M', dob', \theta')$.

From the initial state, we can consider a firing sequence $\sigma \in T^*$ of length n. For $1 \leq i \leq n$, $\sigma(i)$ denotes the i-th transition of the sequence. A sequence σ is a *timed sequential execution* if there exist a set of states and dates $\{M_i, dob_i, \theta_i\}_{1 \leq i \leq n}$ such that for all $1 \leq i \leq n$, $(M_{i-1}, dob_{i-1}, \theta_{i-1}) \xrightarrow{\sigma(i)} (M_i, dob_i, \theta_i)$.

One can see on Figure 1 an example of such firing sequence.

For a given sequence of transitions σ, it is possible te represent symbolically the set of all possible firing dates of the transitions. For each occurrence of transition $\sigma(i)$, θ_i is interpreted as a variable taking its values in $\mathbb{Q}$ and denoting the possible firing dates. The enabling condition is a linear max-plus expression given by $Enabled(M_i, dob_i, \sigma(i), \theta_i)$ (see the example of Figure 1).

3 Concurrent Semantics

3.1 Standard Processes

The first idea is to consider the processes of the underlying untimed net and to select only those that are consistently dated [1].

We use the representation of Engelfriet [6]. Each process is a set E of *events*. We denote $E_\bot \stackrel{\text{def}}{=} E \cup \{\bot\}$ the set E augmented with a special event initial event, denoted $\bot$, which is considered as the origin of the processes. Each event $e \in E$ is a triple (B, t, θ) that represents an occurrence of the transition t (denoted $\tau(e)$) in the process. B is a set of pairs $b \stackrel{\text{def}}{=} (f, p) \in E_\bot \times P$. Such a pair is called a *condition* and refers to the token that has been created by the event f (denoted ${}^\bullet b$) in the place p (denoted $place(b)$). We denote $Place(B) \stackrel{\text{def}}{=} \{place(b) \mid b \in B\}$. When the restriction of $place$ to B is injective, we denote $place^{-1}_{|B}$ its inverse, and for all $P \subseteq Place(B), Place^{-1}_{|B}(P) \stackrel{\text{def}}{=} \{place^{-1}_{|B}(p) \mid p \in P\}$. We denote ${}^\bullet e \stackrel{\text{def}}{=} Place^{-1}_{|B}({}^\bullet t)$ the conditions of B that are consumed and $\underline{e} \stackrel{\text{def}}{=} B \setminus {}^\bullet e$ the conditions that are only read by the event e (represented by read arcs in the process). The set $\{(e, p) \mid p \in \tau(e)^\bullet\}$ of conditions that are created by e is denoted by $e^\bullet$. θ is the date of occurrence of the event (denoted $\theta(e)$). For the initial event $\bot$, we set $\tau(\bot) \stackrel{\text{def}}{=} -, {}^\bullet\bot \stackrel{\text{def}}{=} \emptyset$ and $\bot^\bullet \stackrel{\text{def}}{=} \{(\bot, p) \mid p \in M_0\}$.

The set of final conditions of a process E is $\uparrow E \stackrel{\text{def}}{=} \bigcup_{e \in E_\bot} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e$.

Given a safe Time Petri Net, we can easily define a mapping Π from its firing sequences to their partial order representation as processes (the set E).

Definition 5. (time process) *Let us consider a firing sequence* $\sigma = \{(M_{i-1}, dob_{i-1}, \theta_{i-1}) \xrightarrow{\sigma(i)} (M_i, dob_i, \theta_i)\}_{1 \le i \le n+1}$, *$\Pi$ is defined inductively as follows:*

- $\Pi(\emptyset) \stackrel{\text{def}}{=} \emptyset$,
- $\Pi(\{(M_{i-1}, \theta_{i-1}, lrd_{i-1}) \xrightarrow{\sigma(i)} (M_i, dob_i, \theta_i)\}_{1 \le i \le n+1}) \stackrel{\text{def}}{=}$
  $\Pi(\{(M_{i-1}, dob_{i-1}, \theta_{i-1}) \xrightarrow{\sigma(i)} (M_i, dob_i, \theta_i)\}_{1 \le i \le n}) \cup$
  $\{(Place^{-1}_{|\uparrow E}({}^\bullet\sigma(n+1)), \sigma(n+1), \theta_{n+1})\}$.

Figure 2 shows an example of a process of the Time Petri Net of Figure 1.

As it is done in the context of untimed Petri net, we can define the *unfolding* of a Time Petri Net as the union of all the processes. We denote this (infinite) set $\mathcal{U}$.

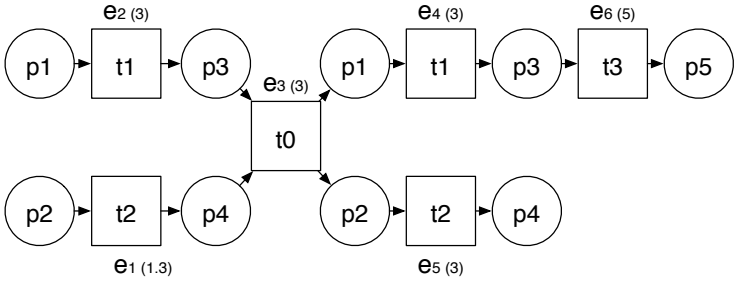


Figure 2. Considering the firing sequence $t_2.t_1.t_0.t_1.t_2.t_3$, the corresponding process is $\{e_1, e_2, e_3, e_4, e_5, e_6\}$, where $e_1 = (\{(\bot, p_2)\}, t_2, 1.3), e_2 = (\{(\bot, p_1)\}, t_1, 3), e_3 = (\{(e_1, p_3), (e_2, p_4)\}, t_0, 3), e_4 = (\{(e_3, p_1)\}, t_1, 3), e_5 = (\{(e_3, p_2)\}, t_2, 3), e_6 = (\{(e_4, p_3)\}, t_3, 5)$.

3.2 Problem with the Superimposition of Processes

Let us consider for example the firing sequences $t_1.t_3$ and $t_1.t_2.t_3$. For $1 \leq i \leq 3$, denote θ_i the variable containing the possible firing dates of t_i. We obtain thus two different constraints for firing t_3: $[\theta_1 \leq 2] \wedge [\theta_1 + 2 \leq \theta_3 \leq \theta_1 + 2]$ and $[\theta_1 \leq 2] \wedge [1 \leq \theta_2 \leq 2] \wedge [\theta_2 \leq \theta_1 + 2] \wedge [\theta_1 + 2 \leq \theta_3 \leq \theta_1 + 2] \wedge [\theta_3 \leq max(\theta_1, \theta_2)]$. The superimposition will define a disjunction of these constraints. This leads to difficult questions when we want to extract the processes from the unfolding.

The main problem with unfoldings of Time Petri Net is to take *urgency* into account. Urgency can prevent the system from staying a given state. It is due to the maximum delays on the transitions. Unfortunately, checking that the maximum delays are not overtaken is not a local computation and requires to check other transitions, possibly in the entire net. We can circumvent the problem by dealing with a simple subclass of Time Petri Net, that we called *Time Extended Free Choice*.

A Time Petri Net is time extended free choice if:

$$\forall t, t' \in T \ \left\{ \begin{array}{l} lfd(t) < \infty \\ {}^\bullet t \cap {}^\bullet t' \neq \emptyset \end{array} \right\} \Rightarrow {}^\bullet t \subseteq {}^\bullet t'$$

In the same spirit, in [10], Semenov and Yakovlev defined a class of time independent choice nets that takes into account not only structural, but also semantical aspects.

To deal with the general case, our approach is to duplicate events with different constraints and to code in the graphical structure the set of events that are the structural and temporal causes. This will permit, as in untimed cased, the extraction of processes in exploring only the predecessors of an event, and thus benefiting of the local aspects of concurrency. We will also show that the unfolding, defined as the union of the time processes, can be finitely represented by a prefix, using a symbolic notion of cutoff event. Figure 3 shows the prefix of our example of Figure 1.

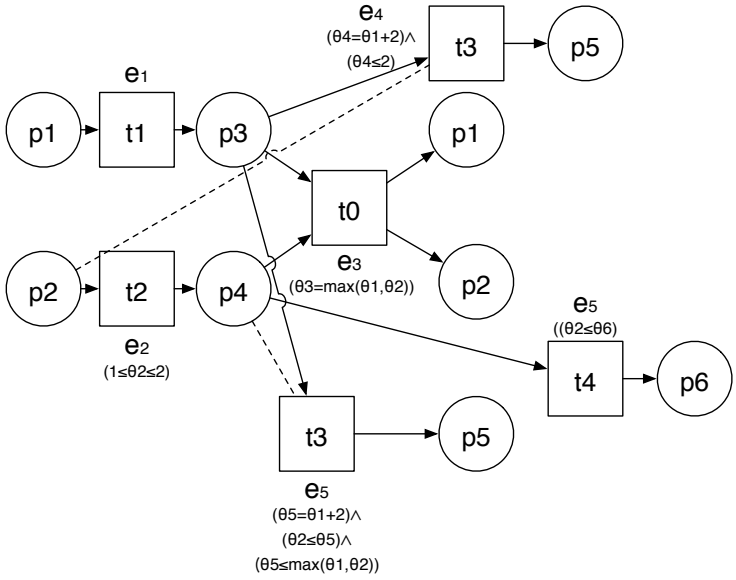


Figure 3. The complete finite prefix of the unfolding of the example of Figure 1. Read arc are depicted using dash lines.

4 Local Firing

At first glance, Time Petri Nets seem intrinsically sequential since the semantics requires to check time conditions for all the enabled transitions in the net. At least, it seems required to know the global states of the net.

In fact, there are cases in which the decision to fire a transition can be taken locally (considering the input places of the transition and its time interval) as in simple Petri nets. The question is to know whether a transition can fire at a given date θ, even if other transitions will fire before θ in other parts of the net. For example, consider the net of Figure 1, starting at time 0 in the marking $\{p_1, p_2\}$. The usual sequential semantics forbids to fire t_1 at time 10 from the initial state because t_2 is enabled and must fire before time 2. However we are allowed to run the net until time 10 without firing t_1 (because its *lfd* is infinite). Whatever has occurred until time 10, nothing can prevent t_1 from firing at date 10, t_1 being the only transition that can remove the token in place p_1.

In contrast, the firing of transition t_3 highly depends on the firing date of transition t_2 (because t_0 fires immediately after its enabling and disables t_3). So, in order to fire t_3, we have to check whether p_2 or p_4 is marked.

This remark leads us to define a *concurrent operational semantics* in which it is possible to fire a transition without knowing the whole marking of the net, but only a partial marking made of the consumed tokens plus possibly some tokens that are only read (not consumed) in order to get enough information.

4.1 Local states

For the sake of simplicity, from now on, we assume that we know a partition $\{P_i\}$ of the set P of places of the net such that places are exclusive: for all reachable marking M, $P_i \cap M$ is a singleton. For a place $p \in P_i$, we denote $\overline{p} \stackrel{\text{def}}{=} P_i \setminus \{p\}$. It is well-known that any net can be partitioned in such a way, with the help of the addition of complementary places. In the example of Figure 1, we can consider the partition $\{\{p_1, p_3, p_5\}, \{p_2, p_4, p_6\}\}$. This partition will be used to test the absence of tokens. In our example, if we want to fire t_3, we have to check that t_0 will not fire before t_3, removing then the token in place p_3. If we know that p_2 is marked, we can deduce that p_4 is not, and thus that t_0 is disabled.

Definition 6. (partial marking) *A set of places $L \subseteq P$ is* complete *if it contains one place per set of mutually exclusive places (the considered partition). In the general case, we will say that it is a* partial marking.

Definition 7. (local state) *A* local state *of a Time Petri Net is a triple (L, dob, lrd) where $L \subseteq P$ is a partial marking, and $dob, lrd :\longrightarrow \mathbb{Q}$ associates a* date of birth *$dob(p)$ and a* latest reading date *$lrd(p)$ with each place $p \in L$.*

Notice that the notion of date of a global state has been replaced by a function acting on places. Local states with complete markings (called *maximal states* in the sequel) will play the role of global states in the standard semantics. It will be said *consistent* if the minimum ages of tokens in the state is less than the latest firing date of the enabled transitions.

Definition 8. (consistent state) *A maximal state (M, dob, lrd) is* consistent *if $\forall t \in T,\ {}^\bullet t \subseteq M \Rightarrow min_{p \in {}^\bullet t}(max_{p' \in M} lrd(p') - dob(p)) \leq lfd(t)$.*

4.2 Local enabling

We propose to relax the enabling condition by considering this one:

$$Local\_enabled(L, dob, t, \theta) = \begin{cases} {}^\bullet t \subseteq L \wedge [\theta \geq max_{p \in {}^\bullet t} dob(p) + efd(t)] \wedge \\ \bigwedge_{t' \in T, {}^\bullet t' \cap L \neq \emptyset} \begin{cases} \bigvee_{p \in {}^\bullet t'} (\overline{p} \cap L \neq \emptyset) \vee \\ \theta \leq max_{p \in {}^\bullet t'} dob(p) + lfd(t') \end{cases} \end{cases}$$

which is clearly weaker.

It is also clear that we augment the flexibility in taking L as smaller as possible. This is why we consider the *minimum enabling condition.*

Definition 9. (minimum local enabling)

$$Min\_local\_enabled(L, dob, t, \theta) \quad \textit{iff} \quad \begin{cases} Local\_enabled(L, dob, t, \theta) \wedge \\ \nexists L' \subset L \ \ Local\_enabled(L', dob_{|L'}, t, \theta) \end{cases}$$

where $dob_{|L'}$ denotes the function dob restricted to the definition domain L'.

4.3 Local semantics

Definition 10. (local firing rule) *The net starts in the* initial maximal state (M_0, dob_0, lrd_0) *where* $lrd_0 \stackrel{\text{def}}{=} dob_0$. *A transition* t *can fire at date* θ *using the partial marking* $L \subseteq M$ *from a maximal state* (M, dob, lrd) *if* $Min\_local\_enabled(L, dob, t, \theta)$ *and for all* $p \in L$, $\theta \geq lrd(p)$. *This action leads to the maximal state* $((M \setminus {}^\bullet t) \cup t^\bullet, dob', lrd')$ *with*

$$dob'(p) \stackrel{\text{def}}{=} \begin{cases} dob(p) & \text{if } p \in M \setminus {}^\bullet t \\ \theta & \text{if } p \in {}^\bullet t \end{cases}$$

and

$$lrd'(p) \stackrel{\text{def}}{=} \begin{cases} lrd(p) & \text{if } p \in M \setminus L \\ \theta & \text{if } p \in (L \setminus {}^\bullet t) \cup t^\bullet \end{cases}$$

The fact that a transition t has been fired using this local rule is denoted $(M, dob, lrd) \xrightarrow{t,\theta,L} (M', dob', lrd')$.

From the initial state, we can consider a firing sequence $\sigma \in T^*$ of length n. A sequence σ is a *local timed sequential execution* if there exist a set of states, dates and partial markings $\{M_i, dob_i, lrd_i, \theta_i, L_i\}_{1 \leq i \leq n}$ such that for all $1 \leq i \leq n$, $(M_{i-1}, dob_{i-1}, lrd_{i-1}) \xrightarrow{\sigma(i),\theta_i,L_i} (M_i, dob_i, lrd_i)$.

5 Time Processes

Given a safe Time Petri Net, we can easily define a mapping Π from its firing sequences to their partial order representation as processes (the set E).

Definition 11. (time process) *Let us consider a local firing sequence* $\sigma = \{(M_{i-1}, dob_{i-1}, lrd_{i-1}) \xrightarrow{\sigma(i),\theta_i,L_i} (M_i, dob_i, lrd_i)\}_{1 \leq i \leq n+1}$, Π *is defined inductively as follows:*

- $\Pi(\emptyset) \stackrel{\text{def}}{=} \emptyset$,
- $\Pi(\{(M_{i-1}, dob_{i-1}, lrd_{i-1}) \xrightarrow{\sigma(i),\theta_i,L_i} (M_i, dob_i, lrd_i)\}_{1 \leq i \leq n+1}) \stackrel{\text{def}}{=} \Pi(\{(M_{i-1}, dob_{i-1}, lrd_{i-1}) \xrightarrow{\sigma(i),\theta_i,L_i} (M_i, dob_i, lrd_i)\}_{1 \leq i \leq n}) \cup \{Place^{-1}_{|\uparrow E}(L_{n+1}), \sigma(n+1), \theta_{n+1}\}$.

Figure 4 shows an example of a process of the Time Petri Net of Figure 1.

What is called the *time unfolding* is just the union of all the processes and is denoted $\mathcal{U}_T$

A process defines strong and weak causalities between its events.

Definition 12. (causality) *Given two events* e *and* f *of a process* E,

- e *strongly causally precedes* f *(written* $e \rightarrow f$*) iff* $e^\bullet \cap ({}^\bullet f \cup \underline{f}) \neq \emptyset$
- e *weakly causally precedes* f *(written* $e \rightsquigarrow f$*) iff* $(e \rightarrow f) \vee (\underline{e} \cap {}^\bullet f \neq \emptyset)$

The strong causal past of an event is denoted by $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$.

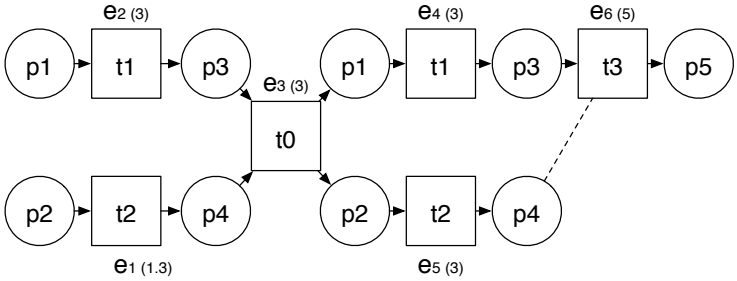


Figure 4. The corresponding time process for the firing sequence $t_2.t_1.t_0.t_1.t_2.t_3$. Notice the read arc introduced by the local semantics.

Now, we prove that processes built with the $Min\_local\_enabled$ condition are correct and complete. Correction means that a time process can always be extended to obtain a standard process of the net (without read arcs).

Each event e of a time process can be mapped to the corresponding event of a standard process defined as:

$$h(e) \stackrel{\text{def}}{=} (\{(h(f), p) \mid (f, p) \in {}^\bullet e\}, \tau(e))$$

This definition is also valid for the initial event $h(\bot) = \bot$. We denote $h(E) \stackrel{\text{def}}{=} \{(h(e), \theta) \mid (e, \theta) \in E\}$

Let us consider a time process E. Let us denote $dob_E(p) \stackrel{\text{def}}{=} \theta({}^\bullet place_{|\uparrow E}{}^{-1}(p))$ the date of birth of the token which stands in place p after the execution of the process E. Additionally we define the latest date when the token which stands in place p was read as: $lrd_E(p) \stackrel{\text{def}}{=} max(dob_e(p), max_{e \in E, b \in \underline{e}} \theta(e))$ (remark that $lrd_E(p) = dob_E(p)$ when no event reads the token). The maximal state reached after the execution of the process E is $End_E \stackrel{\text{def}}{=} (Place(\uparrow E), dob_E, lrd_E)$.

Definition 13. (temporally complete time process) *We will say that a time process E is temporally complete if the maximal state reached after the execution of the process is temporally consistent, i.e.:*

$$\forall t \in T, {}^\bullet t \subseteq Place(\uparrow E) \Rightarrow min_{p \in {}^\bullet t}(max_{p' \in Place(\uparrow E)} lrd_E(p') - dob_E(p)) \leq lfd(t)$$

.

Lemma 1. (the temporally complete processes are the standard processes) $\forall E \in \mathcal{U}_T$, $h(E) \in \mathcal{U}$ *iff E is temporally complete.*

Theorem 1. (correctness) *Our construction of time processes using the $Min\_local\_enabled$ firing condition is correct, which means that for all time processes there exists a standard process E' such that $h(E) \subseteq E'$*

Theorem 2. (completeness) *Our construction of time processes using the Min_local_enabled firing condition is complete, which means that for all standard processes E of the net, there exists a time process E' such that $h(E') = E$.*

We can remark that the unfolding of time extended free choice Petri nets does not contain any read arc. The unfolding is just the union of standard processes.

As a consequence we can prove that a time net with all its time intervals set to $[0, \infty[$ has an unfolding, which is equal to the classical unfolding of the underlying untimed net. Another interesting fact is that considering the unfolding of a Time Petri Net formed with several disconnected paths will never try to connect these parts. As a consequence, the unfolding of the whole net will be exactly the set union of the disconnected parts.

6 Symbolic Unfolding and Complete Finite Prefix

6.1 Process extraction

Despite the superimposition of the time processes, and thus the mixing of all the events belonging to different processes, it is possible to recover the processes. More precisely, given a set of events, we can check whether they form a process or not.

Theorem 3. (process extraction) *Let E a finite set of dated events. E is a time process iff:*

$$\begin{array}{lll} \lceil E \rceil = E & \wedge & (E \textit{ is causally closed}) \\ \nexists e, e' \in E \;\; e \neq e' \wedge {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \wedge & (E \textit{ is conflict free}) \\ \nexists e_0, e_1, ..., e_n \in E \;\; e_0 \rightsquigarrow e_1 \rightsquigarrow ... \rightsquigarrow e_n \rightsquigarrow e_0 & \wedge & (\rightsquigarrow \textit{ is acyclic}) \\ \forall e, e' \in E \;\; e \rightsquigarrow e' \Rightarrow \theta(e) \leq \theta(e') & \wedge & (\theta \textit{ is compatible with } \rightsquigarrow) \end{array}$$

$$\forall e = (B, t) \in E \;\; Min\_local\_enabled(Place(B), dob_{E|\lceil B \rceil}, t, \theta(e))$$

6.2 Complete Finite Prefixes

It has already been shown in [8] that the age of tokens can be reduced to bounded values without losing information about the possible actions in the future.

Definition 14. (reduced age of tokens) *The* reduced age *of a token in a place $p \in M$ in the maximal state $S \stackrel{\text{def}}{=} (M, dob, lrd)$ is*

$$A_S(p) \stackrel{\text{def}}{=} min(max_{p' \in M} lrd(p') - dob(p), max(b(t) \mid t \in T \wedge p \in {}^\bullet t))$$

where $b(t) \stackrel{\text{def}}{=} efd(t)$ if $lfd(t) = \infty$, $lfd(t)$ otherwise.

Two maximal states $S_1 = (M_1, dob_1, lrd_1)$ and $S_2 = (M_2, dob_2, lrd_2)$ will be considered as equivalent (written $S_1 \equiv S_2$) if they have the same marking and same reduced age ($A_{S_1} = A_{S_2}$).

The reduced ages of maximal states can be represented by a symbolic expression.

Theorem 4. (finite number of reduced ages) *The set of expressions A_{End_E} for all the time processes E is finite.*

Theorem 5. (firing a transition from two equivalent consistent states) *Let S_1 and S_2 two equivalent consistent states. Let M their common marking. A transition t can fire from S_1 at date $\theta_1 \geq max_{p\in M} lrd_1(p)$ using the partial marking $L \subseteq M$ iff it can fire from S_2 at date $\theta_2 \stackrel{\text{def}}{=} \theta_1 - max_{p\in M} lrd_1(p) + max_{p\in M} lrd_2(p)$ using the same partial marking L.*

Definition 15. finite processes and complete finite prefix *We define the set of finite time processes FTP from local firing sequences σ as:*

$$E \in FTP \ \ \textit{iff} \ \ (\Pi(\sigma) = E) \wedge (\nexists \sigma' \ \ | \ \sigma' \ |<| \ \sigma \ | \wedge End_{\Pi(\sigma')} \equiv End_{\Pi(\sigma)}).$$

The finite complete prefix *is the union of all the events that appear in the finite complete time processes.*

Remark: this definition could be improved considering the implication of expressions instead of equivalence. It seems also that the generic notion of adequate order is usable in our framework.

6.3 Substitution

To prove the completeness of our notion of prefix, we show that every complete time process can be obtained by substitution of prefixes in processes belonging to the prefix. Knowing that the same actions are possible from equivalent consistent states, reached by two different complete processes, we can translate any continuation of one process to the other providing we also translate the firing dates of the events.

We will say that a complete time process E can be continued to E' (written $E \sqsubseteq E'$) if there exist sequences of local firings σ and σ' such that $\Pi(\sigma) = E$ and $\Pi(\sigma') = E'$ and σ is a prefix of σ'.

Definition 16. (substitution of prefixes) *Let E_1 and E_2 two complete processes, and $E'_2 \sqsubseteq E_2$ such that $End_{E'_2} \equiv End_{E_1}$. The* substitution *operation replaces E'_2 by E_1 in E_2 as:*

$$subst(E_1, E'_2, E_2) \stackrel{\text{def}}{=} E_1 \cup \{\phi(e) \mid e \in E_2 \setminus E'_2\}$$

where

$$\forall e \stackrel{\text{def}}{=} (B, t, \xi) \in E_2 \setminus E'_2 \qquad \phi(e) \stackrel{\text{def}}{=} (\psi(B), t, \xi - max_{f\in E'_2}\theta(f) + max_{f\in E_1}\theta(f))$$

$$\forall b \stackrel{\text{def}}{=} (e, p) \in \bigcup_{f\in E_2\setminus E'_2} {}^\bullet f \cup \underline{f} \ \ \psi(b) \stackrel{\text{def}}{=} \begin{cases} (\phi(e), p) & \textit{if } e \notin E'_2 \\ place^{-1}_{\uparrow E_1}(p) & \textit{if } e \in E'_2 \end{cases}$$

The operation can be generalized as follows:

$$subst(E_0, E'_1, E_1, ..., E'_n, E_n) \stackrel{\text{def}}{=} subst(subst(E_0, E'_1, E_1, ..., E'_{n-1}, E_{n-1}), E'_n, E_n)$$

We can show that complete processes are closed under substitution of prefixes and the following theorem proves the completeness of the prefix construction.

Theorem 6. (extraction of the complete processes from the finite prefix) *For every complete process E, there exist complete finite processes $E_0, ..., E_n$ in FTP and $E'_1 \sqsubseteq E_1, ..., E'_m \sqsubseteq E_m$ such that $E = subst(E_0, E'_1, E_1, ..., E'_m, E_m)$.*

7 Conclusion

We have proposed a definition for the notion of unfolding of Safe Time Petri Nets and its finite representation. The definition is constructive and has been implemented in a prototype. Our prefix construction permits to easily extract time processes and seems to capture graphically a maximum of concurrency. We do not claim that it is the only way to do and we had long discussions with our colleagues about what kind of information must be coded in the graph structure rather than coded in the symbolic constraints. This rises the problem of the evaluation of algorithmic complexities. This is a difficult question that remains to be studied in detail.

Several perspectives are open. From a semantical point of view, this study was faced to the contradiction of having a global time model equiped with a partial order semantics. There is probably some space to imagine models with a more local semantics of time, while keeping good capabilities to specify timed behaviours. On the technical side, the notion of equivalent maximal states can be certainly refined to provide shorter prefixes. One can also think about to deal with with larger class of Petri nets.

References

1. T. Aura and J. Lilius. Time processes for time Petri nets. In *ICATPN*, volume 1248, pages 136–155, 1997.
2. B. Bieber and H. Fleischhack. Model checking of time Petri nets based on partial order semantics. In *CONCUR*, volume 1664 of *LNCS*, pages 210–225, 1999.
3. T. Chatain. Symbolic unfoldings of high-level petri nets and application to supervision of distributed systems. Technical Report 3455, Rennes 1 Univ., dec. 2006.
4. T. Chatain and C. Jard. Time supervision of concurrent systems using symbolic unfoldings of time petri nets. In *FORMATS*, volume 3829 of *LNCS*, pages 193–207, 2005.
5. T. Chatain and C. Jard. Complete finite prefixes of symbolic unfoldings of safe time petri nets. In *ICATPN*, volume 4024 of *LNCS*, pages 125–145, 2006.
6. J. Engelfriet. Branching processes of Petri nets. *Acta Inf.*, 28(6):575–591, 1991.
7. H. Fleischhack and E. Pelz. Hierarchical timed high-level nets and their branching processes. In *ICATPN*, volume 2679 of *LNCS*, pages 397–416, 2003.
8. Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In *ICATPN*, pages 163–181, 2002.

9. P.M. Merlin and D.J. Farber. Recoverability of communication protocols – implications of a theorical study. *IEEE Transactions on Communications*, 24, 1976.
10. A. L. Semenov and A. Yakovlev. Verification of asynchronous circuits using time petri net unfolding. In *DAC*, ACM Press, pages 59–62, 1999.
11. J. Winkowski. Processes of timed Petri nets. *Theoretical Computer Science*, 243(1-2):1–34, 2000.

McMillan's Complete Prefix for Contextual Nets*

Paolo Baldan[1], Andrea Corradini[2], Barbara König[3], and Stefan Schwoon[4]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
[3] Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany
[4] Institut für Informatik (I7), Technische Universität München, Germany

Abstract. In a seminal paper, McMillan proposed a technique for constructing a finite complete prefix of the unfolding of bounded (i.e., finite-state) Petri nets, which can be used for verification purposes. Contextual nets are a generalisation of Petri nets suited to model systems with read-only access to resources. When working with contextual nets, a finite complete prefix can be obtained by applying McMillan's construction to a suitable encoding of the contextual net into an ordinary net. However, it has been observed that if the unfolding is itself a contextual net, then the complete prefix can be significantly smaller than the one obtained with the above technique. A construction for generating such a contextual complete prefix has been proposed for a special class of nets, called read-persistent. In this paper we propose a new algorithm that works for arbitrary semi-weighted, bounded contextual nets. The construction explicitly takes into account the fact that, unlike ordinary or read-persistent nets, an event can have several different histories in contextual net computations.

1 Introduction

In recent years there has been a growing interest in the use of partial-order semantics to deal with the state-explosion problem when model checking concurrent systems. In particular, a thread of research that started with the seminal work by McMillan [10, 11] proposes the use of the *unfolding* semantics as a basis for the verification of finite-state systems, modelled as Petri nets.

The unfolding of a Petri net, originally introduced in [14], is a safe, acyclic *occurrence* net that completely expresses its behaviour. For non-trivial nets the unfolding can be infinite even if the original net is *bounded*, i.e., it has a finite number of reachable states. McMillan's algorithm constructs a *finite complete prefix*, i.e., a subnet of the unfolding such that each marking reachable in the original net corresponds to some concurrent set of places in such a prefix.

* Research partially supported by EU IST-2004-16004 SENSORIA, MIUR Project ART, DFG project SANDS and CRUI/DAAD VIGONI "Models based on Graph Transformation Systems: Analysis and Verification".

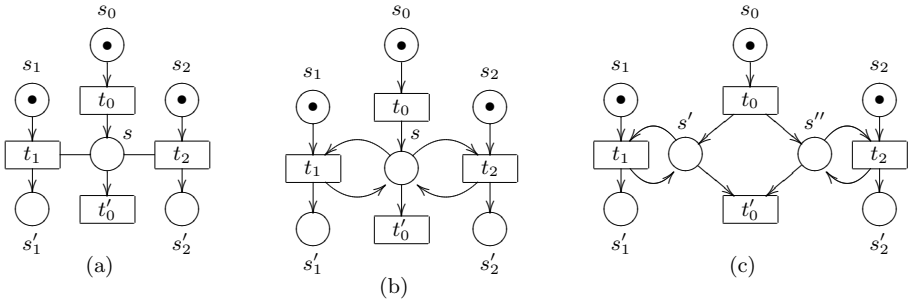


Fig. 1. (a) A safe contextual net; (b) its encoding by replacing read arcs with consume/produce loops; (c) its concurrency-preserving PR-encoding.

Contextual nets [13], also called nets with test arcs [5], activator arcs [8] or read arcs [17], extend ordinary nets with the possibility of checking for the presence of tokens without consuming them. The possibility of faithfully representing concurrent read accesses to resources allows to model in a natural way phenomena like concurrent access to shared data (e.g., reading in a database) [16], to provide concurrent semantics to concurrent constraint programs [12], to model priorities [7] or to conveniently analyse asynchronous circuits [18].

When working with contextual nets, if one is interested only in reachable markings, it is well-known that read arcs can be replaced by consume/produce loops (see Fig. 1(a) and (b)), obtaining an ordinary net with the same reachability graph. However, when one unfolds the net obtained by this transformation, the number of transitions and places might explode due to the sequentialization imposed on readers. A cleverer encoding, proposed in [18] and hereafter referred to as the *place replication encoding* (*PR-encoding*), consists of creating "private" copies of the read places for each reader (see Fig. 1(c)). In this way, for safe nets the encoding does not lead to a loss of concurrency, and thus the explosion of the number of events and places in the unfolding can be mitigated.

A construction that applies to contextual nets and produces an unfolding that is itself a contextual (occurrence) net has been proposed independently by Vogler, Semenov and Yakovlev in [18] and by the first two authors with Montanari in [3]. In particular, the (prefixes of the) unfolding obtained with this construction can be much smaller than in both encodings considered above.

Unfortunately, as discussed in [18], McMillan's construction of the finite complete prefix does not extend straightforwardly to the whole class of contextual nets. The authors of [18] propose a natural generalization of McMillan's algorithm by taking into account some specific features of contextual nets (for example, in the definition of *co-sets*), but they show that their approach only works for contextual nets that are *read-persistent*, i.e., where there is no interference between preconditions and context conditions: any two transitions t_1 and t_2 such that t_1 consumes a token that is read by t_2 cannot be enabled at the same time. Similarly, the algorithm proposed in [2], where McMillan's approach was

generalised to graph grammars, is designed for a restricted class of grammars, which are the graph-grammar-theoretical counterpart of read-persistent nets.

The algorithms of [18] and [2] fail on non-read-persistent systems because, in general, a transition of a contextual occurrence net can have more than one possible *causal history* (or *local configuration*, according to [18]): this happens, for example, when a transition consumes a token which could be read by another transition. In this situation, McMillan's original *cut-off* condition (used by the algorithms in [18] and [2]) is not adequate anymore, because it considers a single causal history for each event (see also the example discussed in Section 3).

In this paper we present a generalization of McMillan's construction that applies to arbitrary bounded *semi-weighted* contextual nets, i.e., Place/Transition contextual nets where the initial marking and the post-set of each transition are sets rather than proper multisets: this class of nets strictly includes safe contextual nets. The proposed algorithm explicitly takes into account the possible histories of events, and generates from a finite bounded semi-weighted contextual net a finite complete prefix of its unfolding. The same constructions and results could have been developed for general weighted contextual nets, at the price of some technical (not conceptual) complications.

As in McMillan's original work, the key concept here is that of a cut-off event, which is, roughly, an event in the unfolding that, together with its causal history, does not contribute to generating new markings. We show that the natural generalisation of cut-off that takes into account all the possible histories of each event is theoretically fine, in the sense that the maximal cut-off free prefix of the unfolding is complete. However, this characterisation is not constructive in general, since an event can have infinitely many histories. We then show how this problem can be solved by restricting the attention to a finite subset of "useful" histories for each event, which really contribute to generating new states.

The contribution of this approach is twofold. From a theoretical point of view, the algorithm extends [18] since it applies uniformly to the full class of contextual nets (and, for read-persistent nets, it specialises to [18]). From a practical point of view, with respect to the approach based on the construction of the complete finite prefix of the PR-encoding, we foresee several improvements. For safe nets the proposed technique produces a smaller unfolding prefix (once the histories recorded for generating the prefix are disregarded) and it has a comparable efficiency (we conjecture that the histories considered when unfolding a safe contextual net exactly correspond to the events obtained by unfolding its PR-encoding). Additionally, our technique appears to be more efficient for non-safe nets (see Appendix A) and it looks sufficiently general to be extended to other formalisms able to model concurrent read accesses to part of the state, like graph transformation systems, for which the encoding approach does not seem viable.

The paper is structured as follows. In Section 2 we introduce contextual nets and their unfolding semantics. In Section 3 we characterise a finite complete prefix of the unfolding for finite-state contextual nets, relying on a generalised notion of cut-off and in Section 4 we describe an algorithm for constructing a complete finite prefix. Finally, in Section 5 we draw some conclusions.

2 Contextual nets and their unfolding

In this section we introduce the basics of marked contextual P/T nets [16, 13] and we review their unfolding semantics as defined in [18, 3].

2.1 Contextual nets

We first recall some notation for multisets. Let A be a set; a *multiset* of A is a function $M : A \to \mathbb{N}$. It is called finite if $\{a \in A : M(a) > 0\}$ is finite. The set of finite multisets of A is denoted by $\mu_* A$. The usual operations on multisets, like multiset union $\oplus$ or multiset difference $\ominus$, are used. We write $M \leq M'$ if $M(a) \leq M'(a)$ for all $a \in A$. If $M \in \mu_* A$, we denote by $[\![M]\!]$ the multiset defined, for all $a \in A$, as $[\![M]\!](a) = 1$ if $M(a) > 0$, and $[\![M]\!](a) = 0$ otherwise. A *multirelation* $f : A \leftrightarrow B$ is a multiset of $A \times B$. It is called *finitary* if $\{b \in B : f(a, b) > 0\}$ is a finite set for all $a \in A$. A finitary multirelation f induces in an obvious way a function $\mu f : \mu_* A \to \mu_* B$, defined as $\mu f(M)(b) = \sum_{a \in A} M(a) \cdot f(a, b)$ for $M \in \mu_* A$ and $b \in B$. In the sequel we will implicitly assume that all multirelations are finitary. A *relation* $r : A \leftrightarrow B$ is a multirelation r where multiplicities are bounded by one, namely $r(a, b) \leq 1$ for all $a \in A$ and $b \in B$. Sometimes we shall write simply $r(a, b)$ instead of $r(a, b) = 1$.

Definition 1 ((marked) contextual net). *A* (marked) contextual Petri net (c-net) *is a tuple* $N = \langle S, T, F, C, m \rangle$*, where*

- S *is a set of* places *and* T *is a set of* transitions*;*
- $F = \langle F_{pre}, F_{post} \rangle$ *is a pair of finitary multirelations* $F_{pre}, F_{post} : T \leftrightarrow S$*;*
- $C : T \leftrightarrow S$ *is a finitary relation, called the* context relation*;*
- $m \in \mu_* S$ *is a finite multiset, called the* initial marking*.*

The c-net is called finite *if* T *and* S *are finite sets. Without loss of generality, we assume* $S \cap T = \emptyset$*. Moreover, we require that for each transition* $t \in T$*, there exists a place* $s \in S$ *such that* $F_{pre}(t, s) > 0$*.*

In the following when considering a c-net N, we will implicitly assume $N = \langle S, T, F, C, m \rangle$.

Given a finite multiset of transitions $A \in \mu_* T$ we write ${}^\bullet A$ for its *pre-set* $\mu F_{pre}(A)$ and $A^\bullet$ for its *post-set* $\mu F_{post}(A)$. Moreover, $\underline{A}$ denotes the *context* of A, defined as $\underline{A} = [\![\mu C(A)]\!]$. The same notation is used to denote the functions from S to the powerset $\mathcal{P}(T)$, e.g., for $s \in S$ we define ${}^\bullet s = \{t \in T : F_{post}(t, s) > 0\}$.

An example of a contextual net, inspired by [18], is depicted in Fig. 2(a). Note that read arcs are drawn as undirected lines. For instance, referring to transition t_1 we have ${}^\bullet t_1 = s_1$, $t_1{}^\bullet = s_3$ and $\underline{t_1} = s_2$.

For a finite multiset of transitions A to be enabled at a marking M, it is sufficient that M contains the pre-set of A and one *additional* token in each place of the context of A. This corresponds to the intuition that a token in a place (like s in Fig. 1(a)) can be used as context concurrently by many transitions; instead, if read arcs are replaced by consume/produce loops (as in Fig. 1(b)) the transitions needing a token in place s can fire only one at a time.

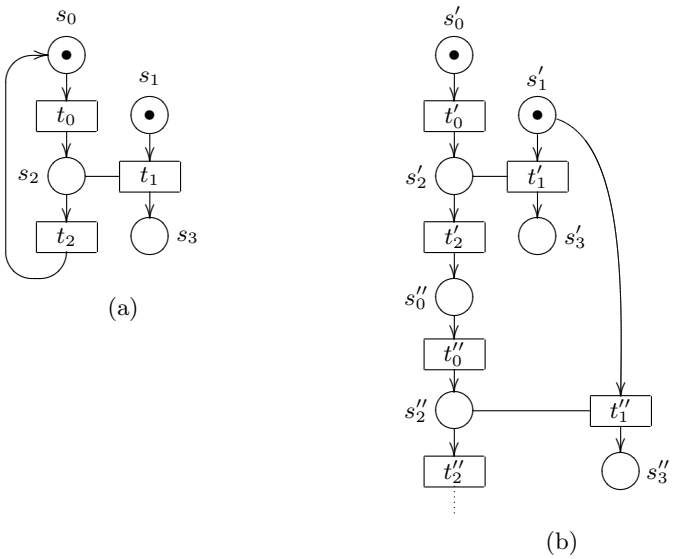


Fig. 2. (a) A contextual net N_0 and (b) its unfolding $\mathcal{U}_a(N_0)$.

Definition 2 (enabling, step). *Let N be a c-net. A finite multiset of transitions $A \in \mu_* T$ is* enabled *at a marking $M \in \mu_* S$ if ${}^\bullet A \oplus \underline{A} \leq M$. In this case, the execution of A in M, called a* step *(or a* firing *when it involves just one transition), produces the new marking $M' = M \ominus {}^\bullet A \oplus A^\bullet$, written as $M\,[A\rangle\,M'$.*

A marking M of a c-net N is called *reachable* if there is a finite sequence of steps leading to M from the initial marking, i.e., $m\,[A_0\rangle\,M_1\,[A_1\rangle\,M_2 \ldots\,[A_n\rangle\,M$.

Definition 3 (bounded, safe and semi-weighted nets). *A c-net N is called* n-bounded *if for any reachable marking M each place contains at most n tokens, namely $M(s) \leq n$ for all $s \in S$. It is called* safe *if it is 1-bounded and F_{pre}, F_{post} are relations (rather than general multirelations). A c-net N is called* semi-weighted *if the initial marking m is a set and F_{post} is a relation.*

Observe that requiring F_{pre} (resp. F_{post}) to be relations amounts to asking that for any transition $t \in T$, the pre-set (resp. post-set) of t is a set, rather than a general multiset.

We recall that considering semi-weighted nets is essential to characterise the unfolding construction, in categorical terms, as a coreflection [4]. However, in this paper, the choice of taking semi-weighted nets rather than general weighted nets is only motivated by the need of simplifying the presentation: the generalisation would require only some technical complications in the definition of the unfolding (Definition 10), related to the fact that an occurrence of a place would not be completely identified by its causal history.

2.2 Occurrence c-nets

Occurrence c-nets are safe c-nets such that the dependency relations among transitions that we will introduce, causality and asymmetric conflict, satisfy suitable acyclicity and well-foundedness requirements.

Causality is defined as for ordinary nets, with an additional clause stating that transition t causes t' if it generates a token in a context place of t'.

Definition 4 (causality). *Let N be a safe c-net. The* causality relation $<_N$ *is the least transitive relation on $S \cup T$ such that*

1. *if $s \in {}^\bullet t$ then $s <_N t$;*
2. *if $s \in t^\bullet$ then $t <_N s$;*
3. *if $t^\bullet \cap \underline{t'} \neq \emptyset$ then $t <_N t'$.*

Given $x \in S \cup T$, we write $\lfloor x \rfloor$ for the set of causes *of x in T, defined as $\lfloor x \rfloor = \{t \in T : t \leq_N x\} \subseteq T$, where $\leq_N$ is the reflexive closure of $<_N$.*

We say that a transition t is in *asymmetric conflict* with t', denoted $t \nearrow_N t'$, if *whenever both t and t' fire in a computation, t fires before t'*. The paradigmatic case is when transition t' consumes a token in the context of t, i.e., when $\underline{t} \cap {}^\bullet t' \neq \emptyset$, as for transitions t'_1 and t'_2 in Fig. 2(b) (see [4, 15, 9, 18]).

Note that the fact that *whenever both t and t' fire, t fires before t'* trivially holds when $t <_N t'$, because t cannot follow t' in a computation, and (with t and t' in interchangeable roles) also when t and t' have a common precondition, since they will never fire in the same computation. For technical convenience the definition of $\nearrow_N$ takes these two situations into account as well.

Definition 5 (asymmetric conflict). *Let N be a safe c-net. The* asymmetric conflict relation $\nearrow_N$ *(also denoted $\nearrow$ if N is clear from the context) is the binary relation on T defined as*

$$t \nearrow_N t' \qquad \textit{iff} \qquad \underline{t} \cap {}^\bullet t' \neq \emptyset \;\; \textit{or} \;\; (t \neq t' \;\wedge\; {}^\bullet t \cap {}^\bullet t' \neq \emptyset) \;\; \textit{or} \;\; t <_N t'.$$

For $X \subseteq T$, $\nearrow_X$ denotes the restriction of $\nearrow_N$ to X, i.e., $\nearrow_X = \nearrow_N \cap (X \times X)$.

An occurrence c-net is a safe c-net that exhibits an acyclic behaviour, satisfying suitable conflict freeness requirements.

Definition 6 (occurrence c-nets). *An* occurrence c-net *is a safe c-net N such that*

- *each place $s \in S$ is in the post-set of at most one transition, i.e. $|{}^\bullet s| \leq 1$;*
- *the* causal relation $<_N$ *is irreflexive and its reflexive closure $\leq_N$ is a partial order, such that $\lfloor t \rfloor$ is finite for any $t \in T$;*
- *the* initial marking *is the set of minimal places w.r.t. $\leq_N$, i.e., $m = \{s \in S : {}^\bullet s = \emptyset\}$;*
- *$\nearrow_{\lfloor t \rfloor}$ is acyclic for all $t \in T$.*

The last condition corresponds to the requirement of irreflexivity for the conflict relation in ordinary occurrence nets. In fact, if a transition t has a $\nearrow_N$ cycle in its causes then it can never fire, since in an occurrence c-net N, the order in which transitions appear in a firing sequence must be compatible with the asymmetric conflict relation. An example of an occurrence c-net can be found in Fig. 2(b).

The notion of concurrency is the natural generalisation of the one for ordinary nets. Note that, because of the presence of contexts, some places that a transition needs in order to fire (the contexts) can be concurrent with the places it produces.

Definition 7 (concurrency relation). *Let N be an occurrence c-net. A finite set of places $M \subseteq S$ is called* concurrent*, written $conc(M)$, if*

1. $\forall s, s' \in M.\ \neg(s < s')$;
2. $\lfloor M \rfloor$ *is conflict-free, i.e.,* $\nearrow_{\lfloor M \rfloor}$ *is acyclic.*

It can be shown that, as for ordinary occurrence nets, a set of places M is concurrent if and only if there is some reachable marking in which all the places of M contain one token.

From now on, consistently with the literature, we shall often call the transitions of an occurrence c-net *events*.

Definition 8 (configuration). *Let N be an occurrence c-net. A set of events $C \subseteq T$ is called a* configuration *if*

1. $\nearrow_C$ *is well-founded;*
2. $\{t' \in C : t' \nearrow t\}$ *is finite for all $t \in C$;*
3. C *is left-closed w.r.t. $\leq$, i.e. for all $t \in C$, $t' \in T$, $t' \leq t$ implies $t' \in C$.*

We denote by $Conf(N)$ the set of all configurations of N, equipped with the ordering defined as $C \sqsubseteq C'$, if $C \subseteq C'$ and $\neg(t' \nearrow t)$ for all $t \in C, t' \in C' \setminus C$.

Furthermore two configurations C_1, C_2 are said to be in conflict *($C_1 \# C_2$) when there is no $C \in Conf(N)$ such that $C_1 \sqsubseteq C$ and $C_2 \sqsubseteq C$.*

The notion of configuration characterises the possible (concurrent) computations of an occurrence c-net. It can be proved that a subset of events C is a configuration iff the events in C can all be fired, starting from the initial marking, in any order compatible with $\nearrow$. The relation $\sqsubseteq$ is a computational order of configurations: $C \sqsubseteq C'$ if C can evolve and become C'. Remarkably, this order is not simply subset inclusion since a configuration C cannot be extended with an event t' if $t' \nearrow t$ for some $t \in C$, since t' cannot fire after t in a computation. Two configurations are in (symmetric) conflict if they do not have a common extension. More concretely $C_1 \# C_2$ when there exists $t_1 \in C_1$ and $t_2 \in C_2 \setminus C_1$ such that $t_2 \nearrow t_1$ or the symmetric condition holds.

Given a configuration C and an event $t \in C$, the *history of t in C* is the set of events that *must* precede t in the (concurrent) computation represented by C. For ordinary nets the history of an event t coincides with the set of causes

$\lfloor t \rfloor$, independently of the configuration where t occurs. Instead, for c-nets, due to the presence of asymmetric conflicts between events, an event t which occurs in more than one configuration may have different histories. The next definition formalises this fact.

Definition 9 (history). *Let N be an occurrence net. Given a configuration C and an event $t \in C$, the* history *of t in C, denoted by $C[\![t]\!]$, is defined as*

$$C[\![t]\!] = \{t' \in C : t'(\nearrow_C)^* t\}.$$

The set of all possible histories of an event t, namely $\{C[\![t]\!] : C \in Conf(N) \wedge t \in C\}$ is denoted by $Hist(t)$.

2.3 Unfolding

Given a semi-weighted c-net N, an *unfolding* construction allows us to obtain an occurrence c-net $\mathcal{U}_a(N)$ that describes the behaviour of N [3, 18]. As for ordinary nets, each event in $\mathcal{U}_a(N)$ represents a particular firing of a transition in N, and places in $\mathcal{U}_a(N)$ represent occurrences of tokens in the places of N. The unfolding is equipped with a mapping to the original net N, sending each place (event) of the unfolding to the corresponding place (transition) in N.

The unfolding can be constructed inductively by starting from the initial marking of N and then by adding, at each step, an occurrence of each transition of N which is enabled by (the image of) a concurrent subset of the places already generated. We present an equivalent axiomatic definition, in the style of the one proposed by Winskel in [20].

Definition 10 (unfolding). *Let $N = \langle S, T, F, C, m\rangle$ be a semi-weighted c-net. The unfolding $\mathcal{U}_a(N) = \langle S', T', F', C', m'\rangle$ of the net N is the unique occurrence c-net satisfying the following (recursive) equations*

$$\begin{array}{l} m' = \{\langle \emptyset, s\rangle : s \in m\} \\ S' = \{m'\} \cup \{\langle t', s\rangle : t' = \langle M_p, M_c, t\rangle \in T' \ \wedge \ s \in t^\bullet\} \\ T' = \{\langle M_p, M_c, t\rangle : M_p, M_c \subseteq S' \ \wedge \ M_p \cap M_c = \emptyset \ \wedge \ conc(M_p \cup M_c) \wedge \\ \qquad t \in T \ \wedge \ \mu f_S(M_p) = {}^\bullet t \ \wedge \ \mu f_S(M_c) = \underline{t}\} \end{array}$$

$$\begin{array}{lll} F'_{pre}(t', s') & \text{iff} & t' = \langle M_p, M_c, t\rangle \ \wedge \ s' \in M_p \quad (t \in T) \\ C'(t', s') & \text{iff} & t' = \langle M_p, M_c, t\rangle \ \wedge \ s' \in M_c \quad (t \in T) \\ F'_{post}(t', s') & \text{iff} & s' = \langle t', s\rangle \quad (s \in S) \end{array}$$

where $f_N = \langle f_T, f_S\rangle : \mathcal{U}_a(N) \to N$ is the folding morphism*, consisting of a pair of mappings $f_T : T' \to T$ and $f_S : S' \to S$ defined by $f_T(t') = t$ for $t' = \langle M_p, M_c, t\rangle$ and $f_S(s') = s$ for $s' = \langle x, s\rangle$.*

As said before, places and events in the unfolding of a c-net represent respectively tokens and firing of transitions in the original net. Each place in the unfolding is a pair recording the "history" of the token and the corresponding place in the

original net. Each event is a triple recording the precondition and context used in the firing, and the corresponding transition in the original net. A new place with empty history $\langle \emptyset, s \rangle$ is generated for each place s in the initial marking. Moreover, a new event $t' = \langle M_p, M_c, t \rangle$ is inserted in the unfolding whenever we can find a concurrent set of places (precondition M_p and context M_c) that corresponds, in the original net, to a marking that enables t. For each place s in the post-set of such t, a new place $\langle t', s \rangle$ is generated, belonging to the post-set of t'. The folding morphism f maps each place (event) of the unfolding to the corresponding place (transition) in the original net.

An initial part of the unfolding of the net N_0 in Fig. 2(a) is represented in Fig. 2(b). The folding morphism from $\mathcal{U}_a(N_0)$ to N_0 is implicitly represented by the name of the items in the unfolding.

The unfolding is complete with respect to the behaviour of the original net in the following sense.

Proposition 1 (completeness of the unfolding). *Let N be a c-net and let $\mathcal{U}_a(N) = \langle S', T', F', C', m' \rangle$ be its unfolding. A marking $M \in \mu_* S$ is coverable in N iff there exists a concurrent subset $X \subseteq S'$ such that $M = \mu f_S(X)$.*

This is the notion of completeness that we will use in the rest of the paper: it is slightly weaker than that of [10, 18], for example, as it is concerned with markings only, and not with transitions.

3 Defining a Complete Finite Prefix

To obtain a finite prefix of the unfolding that is still complete in the sense of Proposition 1, the idea is to avoid to include useless events in the unfolding, where "useless" means events which do not contribute to generating new markings. To this aim McMillan introduced the notion of "cut-off" for ordinary nets, which is roughly an event whose history does not generate a new marking. Then the complete finite prefix is the greatest prefix without cut-offs. This definition of cut-off event has to be adapted to the present framework, since for contextual nets an event may have different histories, or, using McMillan terminology, different local configurations.

Considering only the minimal history of an event, i.e., its set of causes, in the definition of cut-off leads to a finite but not necessarily complete prefix, as observed in [18]. For instance, consider net N_0 in Fig. 2(a). According to the ordinary definition of cut-off, in its unfolding $\mathcal{U}_a(N_0)$ shown in Fig. 2(b) the event t'_2 would be a cut-off since its minimal history $\{t'_0, t'_2\}$ generates a marking corresponding to the initial marking. Thus the largest prefix without cut-offs would be the net O_0 in Fig. 3(a), which is not complete since it does not "represent" the marking $s_0 \oplus s_3$, reachable in N_0.

Considering instead all the possible histories of an event leads to a characterisation of a prefix which is finite and complete, even if this characterisation is not constructive since there can be infinitely many possible histories for a single

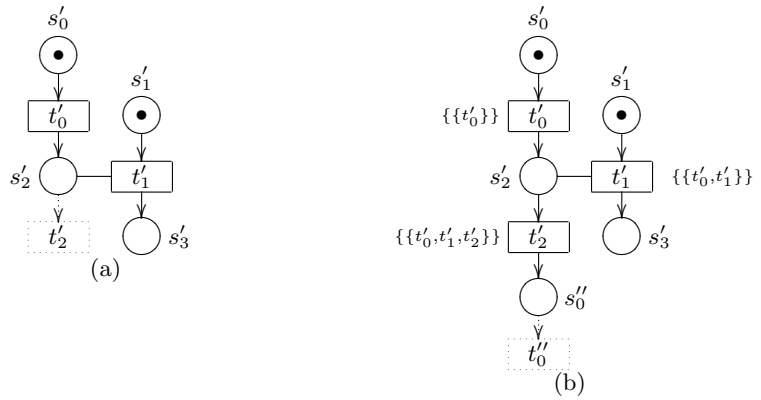


Fig. 3. (a) An incomplete and (b) a complete enriched prefix for the net in Fig. 2.

event (see [2]). In the present paper we suggest to record for each event only a subset of histories which are considered "useful to produce new markings".

To formalise this fact we introduce a notion of occurrence net decorated with possible histories for the involved events.

Definition 11 (enriched occurrence net). *An* enriched occurrence net *is a pair $E = \langle N, \chi \rangle$, where N is an occurrence net and $\chi : T \to \mathcal{P}(\mathcal{P}(T))$ is a function such that for any $t \in T$, $\emptyset \neq \chi(t) \subseteq Hist(t)$.*

The enriched occurrence net E is called closed *if for all $t, t' \in T$, for any $C \in \chi(t)$ if $t' \in C$ then $C[\![t']\!] \in \chi(t')$.*

A configuration of E is a configuration $C \in Conf(N)$ such that $C[\![t]\!] \in \chi(t)$ for all $t \in C$. The set of configurations of E is denoted by $Conf(E)$.

Often, given an enriched occurrence net E we will denote its components by N_E and χ_E. If the enriched net is E_i, we will call its components N_i and χ_i.

A generalisation of the natural prefix ordering over occurrence nets can be defined on enriched occurrence nets.

Definition 12 (prefix ordering). *Given two enriched occurrence nets E_1 and E_2, we say that E_1 is a* prefix *of E_2, written $E_1 \preceq E_2$, if N_1 is a prefix of N_2, and for any $t \in T_1$, $\chi_1(t) \subseteq \chi_2(t)$.*

From now on, $N = \langle S, T, F, C, m \rangle$ is a fixed semi-weighted c-net, $\mathcal{U}_a(N) = \langle S', T', F', C', m' \rangle$ is its unfolding, and $f_N : \mathcal{U}_a(N) \to N$ is the folding morphism.

Definition 13 (enriched event, enriched prefix). *An* enriched event *of the unfolding is a pair $\langle t, H_t \rangle$, where $t \in T'$ is an event of the unfolding, and $H_t \in Hist(t)$ is one of its histories. An* enriched prefix *of the unfolding $\mathcal{U}_a(N)$ is any* closed *enriched occurrence net E such that N_E is a prefix of $\mathcal{U}_a(N)$. We will say that the enriched prefix E contains $\langle t, H_t \rangle$ and write $\langle t, H_t \rangle \in E$ if $t \in T_E$ and $H_t \in \chi_E(t)$.*

An example of enriched prefix of $\mathcal{U}_a(N_0)$ in Fig. 2(b) is given in Fig. 3(b). For any event t the set of histories $\chi_E(t)$ is written near to the event itself.

It can be shown that the set of enriched prefixes of $\mathcal{U}_a(N)$ endowed with the prefix ordering $\preceq$ forms a lattice. Given two enriched prefixes E_1 and E_2, their least upper bound is $E_1 \sqcup E_2 = \langle N_E, \chi_E \rangle$, where N_E is the componentwise union of N_1 and N_2, and, for any event t in N, $\chi_E(t) = \bigcup_{\{i:t\in N_i\}} \chi_i(t)$. Moreover, it is not difficult to prove that given two enriched prefixes E_1 and E_2

$$E_1 \preceq E_2 \qquad \text{iff} \qquad \mathit{Conf}(E_1) \subseteq \mathit{Conf}(E_2).$$

A configuration of $\mathcal{U}_a(N)$ represents a computation in the unfolding itself, which in turn maps, via the folding morphism, to a computation of N. Hence we can define the marking of N after a finite configuration of the unfolding.

Definition 14 (marking after a configuration). *Let $C \in \mathit{Conf}(\mathcal{U}_a(N))$ be a* finite *configuration. We denote by $\mathit{mark}(C)$ the marking of N after C, defined as $\mu f_S(m' \oplus \bigoplus_{t\in C} t^\bullet \ominus \bigoplus_{t\in C} {}^\bullet t)$.*

The notion of cut-off is now defined for enriched events, thus taking histories explicitly into account.

Definition 15 (cut-off). *An enriched event $\langle t, H_t \rangle$ of the unfolding $\mathcal{U}_a(N)$ is called a* cut-off *if either $\mathit{mark}(H_t) = m$, the initial marking of N, or there is another enriched event $\langle t', H_{t'} \rangle$ of $\mathcal{U}_a(N)$ satisfying*

(1) $\mathit{mark}(H_t) = \mathit{mark}(H_{t'})$ and
(2) $|H_{t'}| < |H_t|$.

Let E be an enriched prefix of the unfolding. We say that E contains a cut-off if some enriched event $\langle t, H_t \rangle \in E$ is a cut-off in the full unfolding $\mathcal{U}_a(N)$. The enriched event $\langle t, H_t \rangle \in E$ is called a local cut-off *in E if either $\mathit{mark}(H_t) = m$ or there is an enriched event $\langle t', H_{t'} \rangle \in E$ satisfying (1) and (2) above.*

A different notion of cut-off which refines the one originally proposed by McMillan by using *adequate orders* over configurations has been introduced in [6]. We are confident that this improvement can be integrated seamlessly into our framework, as mentioned in the conclusions.

Note that the notion of cut-off is based on a quantification over all the enriched events of the full unfolding and as such it is not effective. For an enriched event, being a cut-off is a global property, independent of the specific prefix of the unfolding we are considering. Clearly, every local cut-off in an enriched prefix E is also a cut-off. This simple observation will be used several times in the sequel.

Definition 16 (truncation). *The* truncation *$\mathcal{T}_a(N)$ of the unfolding is an enriched occurrence net defined as the greatest enriched prefix (w.r.t. prefix ordering $\preceq$) of the unfolding which does not contain cut-offs.*

The above definition is well-given thanks to the lattice structure of the set of enriched prefixes ordered by $\preceq$. However, it is not yet constructive. In Section 4 we will present an algorithm for computing a complete finite prefix, possibly larger than the truncation, using the notion of local cut-off.

We say that a configuration C of the unfolding includes a cut-off if for some $t \in C$, the enriched event $\langle t, C[\![t]\!]\rangle$ is a cut-off. The next fundamental lemma shows that configurations of the unfolding containing cut-offs can be disregarded without losing information about the reachable markings.

Lemma 1 (cut-off elimination). *Let $C \in Conf(\mathcal{U}_a(N))$ be a finite configuration. There exists a finite configuration C' without cut-offs such that $mark(C) = mark(C')$.*

Using the lemma above we can show that the truncation is a complete prefix of the unfolding.

Theorem 1 (completeness). *The truncation $\mathcal{T}_a(N)$ is a complete prefix of the unfolding, i.e., for any reachable marking M of N there is a finite configuration C of $\mathcal{T}_a(N)$ such that $mark(C) = M$.*

For finite n-bounded nets the number of reachable states of the net is finite and thus one can prove that the truncation of its unfolding is finite. We get this as a corollary of a more general result which will be useful in proving the termination of the algorithm for the complete prefix.

Theorem 2 (finiteness). *Let N be a finite n-bounded c-net and let E be an enriched prefix of the unfolding free of local cut-offs. Then E is finite.*

Recalling that any local cut-off is a cut-off and thus that $\mathcal{T}_a(N)$ is free from local cut-offs we have the following.

Corollary 1. *Let N be a finite n-bounded net. The truncation $\mathcal{T}_a(N)$ is finite.*

For instance, consider the net N_0 and its unfolding $\mathcal{U}_a(N_0)$ in Fig. 2. The truncation $\mathcal{T}_a(N_0)$ is the enriched prefix depicted in Fig. 3(b). Note that it includes the event t'_2. In fact t'_2 has two possible histories: the minimal history $H_2 = \lfloor t'_2 \rfloor = \{t'_0, t'_2\}$ and $H'_2 = \{t'_0, t'_1, t'_2\}$. While $\langle t'_2, H_2\rangle$ is a cut-off, the pair $\langle t'_2, H'_2\rangle$ is not, and thus it is included in the truncation.

4 Computing the prefix

The construction builds incrementally a finite prefix of the full unfolding of a semi-weighted c-net N by starting from the initial marking and by iteratively adding new events representing occurrences of transitions of N. For each event t in *Fin*, the currently built part of the prefix, we also record a current set of histories $\chi_{Fin}(t)$, thus making the prefix under construction an enriched occurrence net. During the construction we record in a set *pe* the enriched events which are

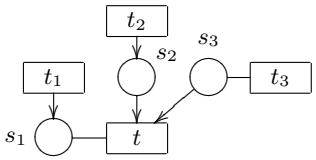


Fig. 4. Predecessors w.r.t. asymmetric conflict of an event t.

candidates for being included in *Fin*, i.e., the pairs $\langle t, H\rangle$ where t is an event enabled in *Fin* and H is one of its current possible histories.

Let us first illustrate how the histories of an event t in a given enriched prefix E can be obtained from the histories of the events that are in direct asymmetric conflict with t. Consider a situation like in Fig. 4, which illustrates a part of the prefix E. A direct predecessor of t w.r.t. asymmetric conflict is either a cause (like t_1, which produces a token that is read, or t_2, which produces a token that is consumed by t) or an event as t_3 that reads a token consumed by t.

A new history for t can be constructed as follows: for every direct cause t_i of t choose any history H_i of t_i, while for every transition t_j that is in direct asymmetric conflict with t (but not a cause) optionally take any history H_j. Whenever such histories are pairwise not in conflict (see Definition 8) then the set $H = \{t\} \cup \bigcup_i H_i$, the union of all such histories (and t), is called a *history for t consistent with E*.

Note that $H \in \mathit{Hist}(t)$ and furthermore adding H to E keeps the prefix closed, since for every transition $t' \in H$ the history $H[\![t']\!]$ is already contained in E. This is a consequence of the fact that for any t_i we have $H[\![t_i]\!] = H_i$ since no two histories in the union are in conflict.

The algorithm proceeds as follows. Again we use the notation of Definition 10.

Initialization: Start with $\mathit{Fin} := m'$ and let χ_{Fin} be the empty function. An event $t = \langle M_p, M_c, \hat{t}\rangle$ is enabled in *Fin* whenever $\mathit{conc}(M_p \cup M_c)$. Now let *pe* be the set of all pairs of the form $\langle t, H_t\rangle$, where t is an event enabled in *Fin* and H_t is a history of t consistent with *Fin*. Initially the only history of t is $\{t\}$.

Loop: While $\mathit{pe} \neq \emptyset$ do: Choose a pair $\langle t, H_t\rangle \in \mathit{pe}$ such that $|H_t|$ is minimal. Remove this pair from *pe*.

- If $\langle t, H_t\rangle$ would be a local cut-off in *Fin*, do nothing.
- If $\langle t, H_t\rangle$ is not a local cut-off, then insert it into *Fin*. This means
  - if t is already present in *Fin* then add the history H_t to $\chi_{\mathit{Fin}}(t)$;
  - otherwise add t to *Fin* and set $\chi_{\mathit{Fin}}(t) := \{H_t\}$.

  Consider all events t' contained either in *Fin* or in *pe*: Whenever t' has a new history $H_{t'}$ consistent with the updated prefix *Fin*, arising from the insertion of H_t, then add $\langle t', H_{t'}\rangle$ to *pe*. (Note that a propagation phase is necessary to obtain all new histories.)

If a new transition has been added to *Fin*, update *pe* by adding all events t which have become enabled in *Fin* in the last step together with all their histories consistent with *Fin*. Then perform the next step of the loop.

Note that whenever a new pair $\langle t', H_{t'} \rangle$ is added to *pe*, then the size of $H_{t'}$ is larger than the size of the history H_t under consideration. This is due to the fact that these newly generated histories must include H_t. Observe also that all pairs $\langle t, H \rangle$ with $H \in \mathit{Hist}(t)$ are considered at some point, unless there exists a local cut-off $\langle t', H' \rangle$ such that $t' \in H$ and $H' = H[\![t']\!]$.

An efficient computation of the prefix should be based on suitable data structures. As observed above, a set of direct predecessors is needed for each event in order to update its histories. Furthermore histories should not be stored explicitly, but via pointer structures containing references back to the histories they originated from. In addition, causality and conflict of histories should be computed incrementally. To this aim it would be helpful to keep trace of all the $\nearrow$-sequences $t_1 \nearrow \ldots \nearrow t_n$ in order to support an easy identification of $\nearrow$-cycles.

It can be shown that at every iteration of the algorithm the prefix *Fin* does not contain local cut-offs. This can be used to prove the correctness and termination of the algorithm.

Theorem 3. *If the net N is finite and n-bounded the algorithm terminates and Fin is complete.*

The complete prefix of a c-net can be much smaller than the complete prefix (constructed using McMillan's algorithm) for the net where read arcs are replaced by consume/produce loops. In fact, consider a net N_1^n analogous to the net in Fig. 1(a) but with n readers $t_1, \ldots, t_n$. Let N_2^n be obtained encoding N_1^n as an ordinary net by simply replacing read arcs with a consume/produce loops, as in Fig. 1(b). The unfolding of net N_2^n includes $k_n = n + n(n-1) + \ldots + n!$ events corresponding to the readers, since each event does not only record the occurrence of a transition, but also its entire history, i.e., the sequence of all events occurring before. Similarly, there are $k_n + 1$ copies of event t'_0. Note that none of these events is a cut-off (according to McMillan's definition), since any two events generating the same marking have histories of equal size. Therefore the complete prefix computed for N_2^n is the unfolding itself. Instead, the complete enriched prefix obtained from N_1^n is the net N_1^n itself, thus it has $n + 2$ transitions only; among them, $t_0, t_1, \ldots, t_n$ have one history each, while t'_0 has 2^n histories. Even if still of exponential size, this prefix is much smaller than the complete prefix of N_2^n, essentially because the order in which the events occurred does not need to be recorded. Moreover, the underlying net obtained by disregarding the histories, which are only auxiliary information needed to construct the prefix, is dramatically smaller in this case.

Now let N_3^n be the PR-encoding of N_1^n, as shown in Fig. 1(c). The unfolding of N_3^n has one occurrence for each of the transitions $t_0, t_1, \ldots, t_n$ and 2^n occurrences of t'_0, none of which is a cut-off (hence, also in this case, the complete prefix

is the full unfolding). Thus there is a one-to-one correspondence between the histories in the enriched prefix of N_1^n and the events of the unfolding of N_3^n. We conjecture that this is a general fact, i.e., the histories of the complete enriched prefix of a safe c-net N are in one-to-one correspondence with the events of the complete finite prefix of the PR-encoding of N. Still, the size of the prefix of N_3^n is exponential in n while the size of the prefix of N_1^n, once the histories are disregarded, is linear.

It is worth stressing that the size of the complete prefixes can be further reduced using adequate orders [6], as remarked also in the conclusion. This would lead to a smaller prefix, for example, for N_2^n.

5 Conclusions

We have presented an approach for computing finite complete prefixes of general contextual nets, which extends the approach proposed for the class of read-persistent nets in [18] and provides an alternative to the technique based on the PR-encoding of contextual nets as ordinary nets. Our work relies on the idea of dealing explicitly with the multiple histories that events can have in contextual net computations, due to the presence of asymmetric conflicts. Subsets of "useful" histories for events are recorded in the prefix during the construction and, correspondingly, a new notion of cut-off is considered. In the case of read-persistent nets every transition has a single history and hence our approach coincides with the one introduced in [18].

Our work shares some basic ideas with [19], where however the definition of cut-off is non-constructive, since it depends on all the possible histories that an event may have. In order to avoid this problem we introduced the (constructive) notion of local cut-off. Apart from that the notion of cut-off in [19] is stronger than ours, which might lead to larger prefixes.

As witnessed by some examples in the paper, the complete prefix of a contextual net can be significantly smaller than that of an equivalent net where read arcs are replaced by consume/produce loops, and it will never be larger. The ability to generate smaller unfoldings comes with a price, i.e., during the construction of the prefix we have to record and evaluate additional information such as histories and asymmetric conflict. Still, we conjecture that the algorithm will never require more space or time than the ordinary algorithm applied to the PR-encoding of the net. More precisely, for safe nets, as discussed in Section 3 the histories in the prefix should correspond exactly to the events in the unfolding of the PR-encoding, and causality and conflict on histories should be the exact match to causality and conflict for transitions. Furthermore we expect our technique to be strictly more efficient for non-safe nets (see Appendix A), since, in this case, the PR-encoding can lead to concurrent occurrences of the same reader where the occurrences share places in consume/produce loops, leading to a blowup in the size of the unfolding.

From a more methodological perspective, let us stress that our approach can build a complete finite prefix for a large class of c-nets directly, without the

need of resorting to an encoding. We think that this feature makes our approach more suitable than others to be extended to other classes of systems exhibiting concurrent read-only accesses, for which an encoding could either not be feasible or could cause a significant loss of concurrency.

In particular, we are interested in graph transformation systems (GTSs), a quite expressive formalism where reading and preserving part of the system state, in this case a graph, is an integral part of the model. We believe that our direct approach will be useful to generalise McMillan's approach to the full class of GTSs, while currently only its read-persistent subclass is dealt with [2]. We are also interested in nets with inhibitor arcs. In this case, an encoding as c-nets would be feasible but it would cause (at least in the non-safe case) a loss of concurrency, and thus a direct approach could be preferable.

We plan to implement and test the algorithm for contextual nets in the framework of the Mole unfolder [1] that currently deals with ordinary nets. At present, with the limited goal of analyzing the size of the produced prefix, we implemented a prototype which given a safe c-net, converts the read arcs into consume/produce loops, builds its finite prefix, and then merges the occurrences of the same context places. A complete implementation of our algorithm is currently in progress. We expect that in order to obtain satisfactory experimental results about the complexity (in time and in space) of our algorithm, in comparison with others, firstly we will need to be able to deal with more refined notions of cut-offs based on adequate orders [6], and secondly we will have to design and implement efficient data structures for recording the sets of histories of an event during the construction of the prefix.

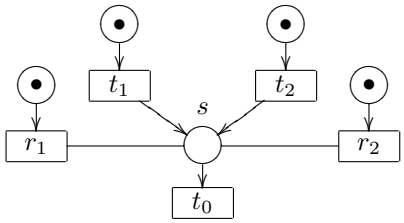
References

1. The Mole unfolder. Available at `http://www.fmi.uni-stuttgart.de/szs/tools/mole`.
2. P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: an unfolding-based approach. In P. Gardner and N. Yoshida, editors, *Proceedings of CONCUR 2004*, volume 3170 of *LNCS*, pages 83–98. Springer Verlag, 2004.
3. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *LNCS*, pages 63–80. Springer Verlag, 1998.
4. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.
5. S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In M. Ajmone-Marsan, editor, *Applications and Theory of Petri Nets*, volume 691 of *LNCS*, pages 186–205. Springer Verlag, 1993.
6. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20:285–310, 2002.
7. R. Janicki and M. Koutny. Invariant semantics of nets with inhibitor arcs. In *Proceedings of CONCUR '91*, volume 527 of *LNCS*. Springer Verlag, 1991.
8. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.
9. R. Langerak. *Transformation and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.

10. K.L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proceedings of CAV '92, Fourth Workshop on Computer-Aided Verification*, volume 663 of *LNCS*, pages 164–174. Springer Verlag, 1992.
11. K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
12. U. Montanari and F. Rossi. Contextual occurrence nets and concurrent constraint programming. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*. Springer Verlag, 1994.
13. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6):545–596, 1995.
14. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
15. G. M. Pinna and A. Poigné. On the nature of events: another perspective in concurrency. *Theoretical Computer Science*, 138(2):425–454, 1995.
16. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets*. PhD thesis, Department of Computer Science - University of Pisa, 1994.
17. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 538–548. Springer Verlag, 1997.
18. W. Vogler, A. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 501–516. Springer Verlag, 1998.
19. J. Winkowski. Reachability in contextual nets. *Fundamenta Informaticae*, 51(1):235–250, 2002.
20. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer Verlag, 1987.

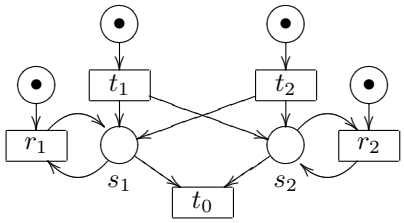
A Unfolding of Non-Safe Nets

Unfolding of a non-safe contextual net N with the algorithm proposed in this paper might lead to an occurrence net smaller than the unfolding of the ordinary net obtained as the PR-encoding of N (see Fig. 1(c)). As an example consider the following net:



The truncation of this net has two occurrences of transition t_0 (either t_0 is caused by t_1 or by t_2), each with four histories (which specify whether r_1 or r_2, or both, or none has been fired before). So in total we have eight histories.

Now consider the corresponding PR-encoding:



Unfolding this net we obtain four occurrences of place s_1 (after firing t_1 or $t_1; r_1$ or t_2 or $t_2; r_1$) and analogously four occurrences of place s_2. All pairs of such places (one representing s_1 and the other s_2) are concurrent. Hence we obtain $4 \cdot 4 = 16$ occurrences of transition t_0.

Intuitively this can be interpreted as follows: the token in s is split into two half-tokens in s_1 and s_2. Then some of the transitions in the unfolding of the encoded net consume "half a token" produced by t_1 and "half a token" produced by t_2.

More generally, consider a net like the one above, but with h writers $t_1, \ldots, t_h$ and k readers $r_1, \ldots, r_k$. Then the truncation of the contextual net has h occurrences of t_0 with a total number of histories $h \cdot 2^k$, since t_0 can consume the token produced by any of the h writers, after it has been read by any subset of the k readers. Instead, the unfolding of the PR-encoding of the net includes $(h \cdot 2)^k$ occurrences of t_0, since each occurrence of t_0 consumes k tokens, and each of these tokens can be produced by any of the h readers and it could have possibly been produced/consumed by the corresponding reader.

Directed Unfolding of Petri Nets

Blai Bonet[1], Patrik Haslum[2], Sarah Hickmott[3], and Sylvie Thiébaux[2]

[1]Departamento de Computación, Universidad Simón Bolívar, Caracas, Venezuela,
[2]National ICT Australia & The Australian National University, Canberra, Australia
[3]National ICT Australia & The University of Adelaide, Adelaide, Australia

Abstract. The key to efficient on-the-fly reachability analysis based on unfolding is to focus the expansion of the finite prefix towards the desired marking. However, current unfolding strategies typically equate to blind (breadth-first) search. They do not exploit the knowledge of the marking that is sought, merely entertaining the hope that the road to it will be short. This paper investigates *directed unfolding*, which exploits problem-specific information in the form of a heuristic function to guide decisions to the desired marking. In the unfolding context, heuristic values are estimates of the distance between configurations. We show that suitable heuristics can be automatically extracted from the original net. We prove that unfolding can rely on heuristic search strategies while preserving the finiteness and completeness of the generated prefix, and in some cases, the optimality of the firing sequence produced. Experimental results demonstrate that directed unfolding scales up to problems that were previously out of reach of the unfolding technique.

1 Introduction

The Petri net unfolding process, originally introduced by McMillan [1], has gained the interest of researchers in verification (see e.g. [2]), diagnosis [3] and, more recently, planning [4]. All have reason to analyse reachability in distributed transition systems, looking to unfolding for some relief of the state explosion problem. Unfolding a Petri net reveals all possible partially ordered runs of the net, without the combinatorial interleaving of independent events. Whilst the unfolding can be infinite, McMillan identified the possibility of a finite prefix with all reachable states. Esparza, Römer and Vogler generalised his approach, to produce the now commonly used ERV unfolding algorithm [5]. This algorithm involves a search, but does not mandate any particular search strategy. Typically, it has been implemented as a breadth-first search, using the length of paths as the primary means to select the next node to add and to determine cut-off events. The MOLE unfolding tool[1] follows this strategy.

Of the various unfolding-based reachability techniques, experimental results indicate on-the-fly analysis to be most efficient for identifying a single, reachable marking [6]. Nevertheless, generating the complete prefix up to a particular state via breadth-first search quickly becomes impractical when the unfolding is wide or the shortest path to the state is deep. It has not been obvious what other

[1] `http://www.fmi.uni-stuttgart.de/szs/tools/mole/`

strategies could be used in the ERV algorithm; recent results have shown depth-first search is incorrect [7]. In this paper, we investigate *directed unfolding*, an approach that takes advantage of information about the sought marking to guide the search. The reason why such an informed strategy has not been considered before may be that unfolding is typically used to prove the absence of deadlocks: this has set the focus on making the entire prefix smaller rather than on reducing the part of the search space explored to reach a particular marking. However, information about the goal marking can be put to good use also in the case when this marking is not reachable.

Inspired by heuristic search in artificial intelligence, particularly in the area of automated planning, directed unfolding exploits problem-specific information in the form of a heuristic function to guide search towards the desired marking. Specifically, heuristics are estimates of the shortest distance from one state to another. Directed unfolding implements a search strategy which explores choices in increasing order of estimated distance to the target marking, as given by the heuristic. If the heuristic is sufficiently informative, this order provides effective guidance towards the marking sought. Whilst the order is not always adequate, in the sense defined in [5], it still guarantees finiteness and completeness of the generated prefix. Interestingly, our proof relies on the observation that adequate orders are stronger than necessary for these purposes, and introduces a weaker notion of semi-adequate ordering.

Techniques for automatically extracting suitable heuristics from the representation of a transition system and using them to guide search, have significantly improved the scalability of automated planning [8–10]. We show that heuristic values can be similarly calculated from a Petri net. If the chosen heuristic is *admissible* (meaning it never overestimates the actual shortest distances) then directed unfolding finds the *shortest* path to the target marking, just like breadth-first search. Using inadmissible heuristics, completeness and correctness are preserved, and performance is often dramatically improved at the expense of optimality. Altogether, directed unfolding can solve much larger problems than the original breadth-first ERV algorithm. Moreover, its implementation requires only minor additions to the latter.

The paper is organised as follows. Section 2 provides an overview of place-transition nets, unfolding, and on-the-fly reachability analysis. Section 3 describes the ideas behind directed unfolding and establishes its theoretical properties. In Section 4, we show how to automatically extract a range of heuristics from the Petri net description. In Section 5 we present experimental results covering Petri net benchmarks and Petri net formalisations of automated planning benchmarks. These show that directed unfolding can provide a significant speed up over breadth-first ERV. Section 6 concludes with remarks about related and future work.

2 Petri Nets, Unfolding and Reachability Analysis

2.1 Place Transition Petri Nets

Petri nets provide a factored representation of discrete-event systems. States are not enumerated and flattened into single nodes, but rather captured by explicit

variable-event relationships. We consider so-called called *place-transition* (PT) nets, and describe them here only briefly; a detailed expose can be found in [11].

A PT-net consists of a net N and its initial marking M_0. The net is a directed bipartite graph where the nodes are places P and transitions T. Typically, places represent the state variables and transitions the events of the underlying discrete-event system. The dynamic behaviour is captured by the flow relation F between places and transitions and vice versa. The *marking* M of a PT-net represents the state of the system. It assigns to each place zero or more tokens.

Definition 1. *A PT-net is a 4-tuple* (P, T, F, M_0) *where* P *and* T *are disjoint finite sets of places and transitions, respectively,* $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ *is a flow relation indicating the presence (1) or absence (0) of arcs, and* $M_0 : P \to \mathbb{N}$ *is the initial marking.*

The *preset* ${}^\bullet x$ of node x is the set $\{y \in P \cup T : F(y, x) = 1\}$, and its *postset* $x^\bullet$ is the set $\{y \in P \cup T : F(x, y) = 1\}$. The marking M enables a transition t if $M(p) > 0$ for all $p \in {}^\bullet t$. The occurrence, or *firing*, of an enabled transition t absorbs a token from each of its preset places and puts one token in each postset place. This corresponds to a state transition in the modeled system, moving the net from M to the new marking M' given by $M'(p) = M(p) - F(p, t) + F(t, p)$ for each p; this is denoted as $M \xrightarrow{t} M'$. In this paper we only consider safe (or 1-safe) nets, meaning it is never possible for more than one token to exist in a place. A *firing sequence* $\sigma = t_1, \ldots t_n$ is a legal sequence of transition firings, i.e. there exists markings $M_1, \ldots, M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1} \xrightarrow{t_n} M_n$. This is noted $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if there exists a firing sequence σ such that $M_0 \xrightarrow{\sigma} M$.

2.2 Unfolding

Unfolding is a method for reachability analysis which exploits and preserves concurrency information in the Petri net. It generates all possible firing sequences of the net, from the initial marking, whilst maintaining a partial order of events based on the causal relation induced by the net.

Unfolding a PT-net produces a pair $\beta = (ON, \varphi)$ where $ON = (B, E, F')$ is an occurrence net, which is a PT-net without cycles, self conflicts nor backward conflicts, and φ is a homomorphism from ON to N that associates the places/transitions of ON with the places/transitions of the PT-net. A node x is in self conflict if there exists two paths to x which start at the same place and immediately diverge. Backward conflict happens when two transitions output to the same place. Such cases are undesirable since in order to decide whether a token can reach a place in backward conflict, it would be necessary to reason with disjunctions such as from which transition the token came from. Therefore, the process of unfolding involves breaking all reachable places in backward conflict by making independent copies of such places, and thus the ON net may contain multiples copies of the places and transitions of the original net which are identified with the homomorphism. In the occurrence net ON, places and transitions are called conditions B and events E respectively. The initial marking M_0 defines a set of initial conditions B_0 in ON such that the places initially marked

are in 1-1 correspondence with the conditions in B_0. The set B_0 constitutes the "seed" of the unfolding.

2.3 Configurations

To understand how an unfolding is built, the most important notions are that of a configuration and local configuration. A *configuration* represents a possible partial run of the net. It is any set of events C such that:

1. C is causally closed: $e \in C \Rightarrow e' \in C$ for all $e' \leq e$,
2. C contains no forward conflict: ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$ for all $e_1 \neq e_2$ in C;

where $e' \leq e$ means there is a path from e' to e in ON. Clearly, a configuration C is a fragment of ON such that all events in C can be ordered into a firing sequence with respect to B_0.

A configuration C can be associated with a marking $\mathrm{Mark}(C)$ of the original PT-net by identifying which conditions will contain a token after the events in C are fired from the initial marking; i.e. $\mathrm{Mark}(C) = \varphi((B_0 \cup C^\bullet) \setminus {}^\bullet C)$ where $C^\bullet$ (resp. ${}^\bullet C$) is the union of postsets (resp. presets) of all events in C. In other words, the marking of C identifies the resultant marking of the original PT-net when (only) the events in C occur. The *local configuration* of an event e, denoted $[e]$ is the minimal configuration containing event e. A set of conditions can be simultaneously marked if the union of the local configurations of their presets forms a configuration.

2.4 Finite Complete Prefix

The unfolding process involves identifying which transitions are enabled by those conditions, currently in the occurrence net, that can be simultaneously marked. These are referred to as the possible next events. A new instance of each is added to the occurrence net, as are instances of the places in their postsets.

The unfolding process starts from the seed B_0 and extends it iteratively. In most cases, the unfolding β is infinite and thus cannot be built. However, it is not necessary to build the complete unfolding β, but only a *complete finite prefix* β' of β that contains all the information in β. Formally, a prefix β' of β is *complete* if for every reachable marking M, there exists a configuration $C \in \beta'$ such that $\mathrm{Mark}(C) = M$, and for every transition t enabled by M, there is a configuration $C \cup \{e\}$ such that $e \notin C$ and $\varphi(e) = t$.

The key to obtaining a complete finite prefix is to identify those events at which we can cease unfolding without loss of information. Such events are referred to as *cut-off events* and can be defined in terms of an *adequate order* on configurations [1, 5]. In the following, $C \oplus \mathcal{E}$ denotes a configuration that extends C with the finite set of events $\mathcal{E}$ disjoint from C.

Definition 2. *A partial order $\prec$ on finite configurations is adequate if*

(a) $\prec$ is well founded, i.e. it has no infinite descending chains,
(b) $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$, and

Algorithm 1 The ERV Unfolding Algorithm

Add the conditions in B_0 to the prefix β
Initialise the priority queue with the events possible in B_0
Note: the queue is sorted in increasing order wrt $\prec$
while the queue is not empty:
    Remove the first event e in the queue (minimal with respect to $\prec$)
    if e is not identified as a cut-off, i.e. if $\nexists e' \in \beta$ s.t. $[e'] \prec [e]$ and $\text{Mark}(e) = \text{Mark}(e')$
        Add the event and its postset to β
        Identify the possible next events and insert them in the queue
    endif
endwhile
Add all cut-off events and their postsets to β

(c) *$\prec$ is preserved by finite extensions: if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then for all finite extensions $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$ such that $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic [2], we have $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.*

Without threat to completeness, we can cease unfolding from an event e, if e takes the net to a marking which can be caused by some other event e' such that $[e'] \prec [e]$. This is because the events (and thus marking) which proceed from e will also proceed from e'. Relevant proofs can be found in [5]:

Definition 3. *Let $\prec$ be an adequate partial order. An event e is a cut-off event with respect to $\prec$ if the prefix contains some event e' such that $Mark([e]) = Mark([e'])$ and $[e'] \prec [e]$.*

2.5 ERV Algorithm

MOLE is a freeware program which unfolds 1-safe PT-nets. It builds the complete finite prefix following the ERV algorithm depicted in Algorithm 1. MOLE uses an adequate order $\prec$ on configurations defined by $C \prec C'$ iff $|C| < |C'|$, further refined into a total order which results in minimal complete prefixes being produced [5]. Note that using this order equates to a breadth-first search strategy.

2.6 On-The-Fly Reachability Analysis

We define the *reachability problem* (also often called coverability problem) for 1-safe PT-nets as follows:

> REACHABILITY: Given a PT-net (P, T, F, M_0) and a subset $P' \subseteq P$, determine whether there is a firing sequence σ such that $M_0 \xrightarrow{\sigma} M$ where $M(p) = 1$ for all $p \in P'$.

This problem is PSPACE-complete [13]. There are various unfolding-based algorithms that decide reachability. Some build the complete finite prefix once and

[2] We say $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic if they are *structurally isomorphic* [12].

use it to answer multiple reachability questions with e.g. SAT or linear programming [6]. Note that deciding reachability is still NP-complete in the size of the complete finite prefix [6].

In contrast, we are interested in the on-the-fly approach, which experimental results have shown to be most efficient when considering just a single marking [6]. This method was first suggested by McMillan [1]. It involves introducing a new transition t_R to the original net, such that ${}^\bullet t_R = P'$. The net is then unfolded, as described previously, but stops when an event e_R, such that $\varphi(e_R) = t_R$, is retrieved from the queue.[3] At this point we can conclude the set of places P' is reachable. If no such event is identified, the complete finite prefix will be generated, indicating that P' is not reachable.

3 Directing the Unfolding

3.1 Intuitive Idea

In the context of the REACHABILITY problem, we are only interested in checking whether the transition t_R is reachable. An unfolding algorithm that doesn't use this information is probably not the best approach. In this section, we aim at a *principled method* of using this information for building the finite prefix in order to turn unfolding into an informed algorithm oriented at solving the reachability task. The resulting approach is called "directed unfolding" as opposed to the standard "blind unfolding"[4].

The basic idea is that for solving REACHABILITY, the unfolding process can be understood as a *search process* on the quest for t_R. Thus, when selecting events from the queue, we should favor those events "closer" to t_R as their systematic exploration results in a more efficient search strategy. This approach is only possible if the finite prefix being built is guaranteed to be complete. Otherwise, the algorithm might erroneously conclude that t_R is not reachable. Not every strategy of pulling events out of the queue is sound. Even a simple depth-first unfolding strategy was recently shown to be incorrect in general [7].

We show that the ERV algorithm can be used with the same definition of cut-off events when the notion of adequate orderings is replaced by a weaker notion that we call *semi-adequate* orderings. This is prompted by the observation that Definition 2 is a sufficient but not a necessary condition for a sound definition of cut-off events. Indeed, just replacing condition (b) in Definition 2 by a weaker condition opens the door for a family of semi-adequate orderings that allow us to direct the unfolding process.

3.2 Principles

As is standard in state-based search, our orderings are based upon the values of a function f that maps configurations into non-negative numbers. Such function

[3] If $[e_R]$ is not required to be the shortest possible firing sequence, it is sufficient to stop as soon as e_R is generated as one of the possible next events. To guarantee optimality however, even with breadth-first unfolding, it is imperative to wait until the event is pulled out of the queue.

[4] The term "directed" has been used elsewhere to emphasize the informed nature of other model-checking algorithms [14].

f defines the ordering $\prec_f$ as

$$C \prec_f C' \quad \text{iff} \quad \begin{cases} f(C) < f(C') & \text{if } f(C) < \infty \\ |C| < |C'| & \text{if } f(C) = f(C') = \infty. \end{cases}$$

Furthermore, the function f is composed of two parts: $f(C) = g(C)+h(C)$ where $g(C) = |C|$ is the number of events in C, and $h(C)$ is any non-negative function such that $h(C) = 0$ if $t_R \in C$ and such that for all pairs of configurations C_1 and C_2 $\text{Mark}(C_1) = \text{Mark}(C_2) \Rightarrow h(C_1) = h(C_2)$. Notice that taking $h \equiv 0$ makes $\prec_f$ into the adequate ordering used by McMillan [1], and that furthermore, by breaking ties appropriately, $\prec_f$ becomes the strict adequate ordering defined in [5] and used in the MOLE implementation of the ERV algorithm.

Following the terminology in heuristic search, the g component is referred to as the cost associated to configuration C while h is referred to as the "heuristic", estimated cost, or distance to reach transition t_R from $\text{Mark}(C)$.

Let us define $h^*(C) = |C'| - |C|$ where $C' \supseteq C$ is the configuration of minimum cardinality that contains t_R if one exists, and ∞ otherwise. We then say that h is an *admissible* heuristic if $h(C) \leq h^*(C)$ for all finite configurations C. Likewise, let us say that a finite configuration C^* is *optimal* if $t_R \in C^*$ and C^* is of minimum cardinality among such configurations. We then have,

Theorem 1 (Main). *Let $f(C) = g(C)+h(C)$ and consider the ordering $\prec_f$ as defined above. Then, (i) the ERV algorithm equipped with $\prec_f$ solves* REACHABILITY*, and (ii) it finds an optimal configuration if one exists and h is admissible.*

Optimal configurations are important in the context of diagnosis since they provide shortest firing sequences to reach a given marking, e.g. a faulty state in the system. A consequence of the theorem is that the original ERV algorithm, which equates to taking $h \equiv 0$, finds shortest firing sequences. In the next two sections, we will give examples of heuristic functions, admissible and non-admissible, and experimental results on benchmark problems. In the rest of this section, we provide the technical characterization of semi-adequate orderings and their relation to adequate ones, as well as the proofs required for the main theorem.

3.3 Technical Details

Upon revising the role of adequate orders when building the complete finite prefix, we found that condition (b), i.e. $C \subset C' \Rightarrow C \prec C'$, in Definition 2 is only needed to guarantee the finiteness of the generated prefix. Indeed, let n be the number of reachable markings of the net and consider an infinite sequence of events $e_1 < e_2 < e_3 < \cdots$ in the unfolding. Then, there are $i < j \leq n + 1$ such that $\text{Mark}([e_i]) = \text{Mark}([e_j])$, and since $[e_i] \subset [e_j]$, condition (b) implies $[e_i] \prec [e_j]$ making $[e_j]$ into a cut-off event, and thus the prefix is finite [5]. A similar result can be achieved if condition (b) is replaced by the weaker condition that in every infinite chain $e_1 < e_2 < e_3 < \cdots$ of events there are $i < j$ such that $[e_i] \prec [e_j]$. We thus define

Definition 4. *A partial order $\prec$ on finite configurations is semi-adequate if*

(*a*) $\prec$ *is well founded, i.e. it has no infinite descending chains,*
(*b*) *in every infinite chain* $C_1 \subset C_2 \subset C_3 \subset \cdots$, *there are* $i < j$ *such that* $C_i \prec C_j$, *and*
(*c*) $\prec$ *is preserved by finite extensions: if* $C_1 \prec C_2$ *and* $Mark(C_1) = Mark(C_2)$, *then for all finite extensions* $C_1 \oplus \mathcal{E}_1$ *and* $C_2 \oplus \mathcal{E}_2$ *such that* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *are isomorphic, we have* $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.

Theorem 2 (Finiteness and Completeness). *If* $\prec$ *is semi-adequate, the prefix produced by the ERV algorithm (Algorithm 1) is finite and complete.*

Proof. The completeness proof is identical to the proof of Proposition 4.9 in [5, p. 14] which states the completeness of the prefix computed by ERV for adequate orderings: this proof does not rely on condition (*b*) at all[5]. The finiteness proof is similar to the proof of Proposition 4.8 in [5, p. 13] which states the finitness of the prefix computed by ERV for adequate orderings. That proof has three items: (1) shows that each chain of events in the prefix is finite, (2) shows that for each event in the prefix, its pre- and postset are finite, and (3) shows that there are only finitely many reachable events at each depth of the prefix. The proofs of items (2) and (3) do not rely on condition (*b*) and can be reused verbatim. The proof of (1) can be replaced by a proof by contradiction as follows. Suppose that an infinite chain $e_1 < e_2 < e_3 < \cdots$ of events exists in the prefix. Each event e_i defines a configuration $[e_i]$ with marking $\mathrm{Mark}([e_i])$, and since the number of markings is finite, there is at least one marking that appears infinitely often in the chain. Let $e'_1 < e'_2 < e'_3 < \cdots$ be an infinite subchain such that $\mathrm{Mark}([e_1]) = \mathrm{Mark}([e_j])$ for all $j > 0$. By condition (*b*) of semi-adequate orderings, there are $i < j$ such that $[e_i] \prec [e_j]$ that together with $\mathrm{Mark}([e_i]) = \mathrm{Mark}([e_j])$ implies that e_j is a cut-off event and thus the chain cannot be infinite. □

Clearly, if $\prec$ is an adequate order, then it is a semi-adequate order. The converse is not necessarily true. The fact that $\prec_f$ is semi-adequate follows directly from the observation that $g(C) = |C|$ is a strictly monotone function with respect to set inclusion, i.e. $C \subset C' \Rightarrow g(C) < g(C')$. Additionally, the property that configurations with identical markings have identical h-values is required for condition (c).

Theorem 3 (Semi-Adequacy of $\prec_f$). *For any* g *and* h *satisfying the assumptions,* $\prec_f$ *is a semi-adequate order.*

Proof. The fact that $\prec_f$ is transitive follows directly from its definition. For well-foundedness, let $C_1 \succ_f C_2 \succ_f \cdots$ be an infinite descending chain of *finite* configurations with markings $M_1, M_2, \ldots$ respectively. Clearly, not all C_i can be such that $f(C_i) = \infty$ since, in virtue of the definition of $\prec_f$, this would imply $\infty > |C_1| > |C_2| > \cdots \geq 0$ which is impossible. By a similar argument, only finitely many C_is have infinite f-value. Let $C'_1 \succ_f C'_2 \succ_f \cdots$ be the subchain where $f(C'_i) < \infty$ for all $i > 0$, and $M'_1, M'_2, \ldots$ be the corresponding markings. Since the number of markings is finite, we can extract a further subsubchain $C''_1 \succ_f C''_2 \succ_f \cdots$ such that $\mathrm{Mark}(C''_1) = \mathrm{Mark}(C''_j)$ for all $j > 0$, and thus

[5] Please see note at the end of the paper.

$h(C''_1) = h(C''_j)$ for all $j > 0$. Therefore, $g(C''_1) > g(C''_2) > \cdots \geq 0$ which is impossible since $g(C) = |C|$ and all C''_i's are finite.

For condition (b), let $C_1 \subset C_2 \subset \cdots$ be an inifinite chain of finite configurations with markings $M_1, M_2, \ldots$ respectively. As before, since the number of marking is finite, there is a subchain $C'_1 \subset C'_2 \subset \cdots$ whose markings are all equal, and thus $h(C'_1) = h(C'_j)$ for all $j > 0$. On the other hand, since all configurations are finite and g is strictly monotone, $0 \leq g(C'_1) < g(C'_2) < \infty$. If $h(C'_1) = \infty$, then $C'_1 \prec_f C'_2$, otherwise $f(C'_1) = g(C'_1) + h(C'_1) < g(C'_2) + h(C'_2) = f(C'_2) < \infty$ and $C'_1 \prec_f C'_2$.

Finally, if $C_1 \prec_f C_2$ have same markings and the extensions $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic, the extensions $C'_1 = C_1 \oplus \mathcal{E}_1$ and $C'_2 = C_2 \oplus \mathcal{E}_2$ also have the same markings, and is straightforward to show that $C'_1 \prec_f C'_2$. □

We are now in a position to prove the main theorem. The fact that ERV equipped with $\prec_f$ solves Reachability follows directly from Theorems 2 and 3. It remains to show that if h is admissible and t_R is reachable, then ERV finds an *optimal* configuration.

Proof (of Theorem 1 (ii)). For a proof by contradiction, assume that the configuration $[e_R]$ for the first event e_R found by ERV is not optimal. Observe that the queue always contains an event e such that $[e]$ is a prefix of an optimal configuration C^* (by induction since it holds at the beginning and remains so after each iteration of ERV). Let e be such an event in the queue when ERV pulls e_R out of the queue. We have that

$$f([e]) \;=\; g([e]) + h([e]) \;\leq\; |[e]| + h^*([e]) \;=\; |[e]| + |C^*| - |[e]| \;=\; |C^*|$$

since $C^* \supseteq [e]$ is the smallest configuration containing t_R. On the other hand, $[e_R]$ being non-optimal, $f([e_R]) = |[e_R]| > |C^*|$. Thus, $f([e_R]) > f([e])$ and so e_R could not have been pulled out of the queue. □

A final remark will help complete the picture of the relationship between adequate orderings, semi-adequate ones, and properties of heuristic functions. Hickmott *et. al* [4] consider a stronger property of heuristics called *monotonicity*. h is monotone iff it satisfies the triangle inequality $h(C) \leq |C'| - |C| + h(C')$ for all finite $C' \supseteq C$. Monotonicity implies admissibility. The converse is not true, but in practice, it is difficult to automatically extract good admissible heuristics which are not monotone. In fact, all admissible heuristics used in automated planning are monotone. Hickmott *et. al* [4] show that if h is monotone, then $\prec_f$ becomes adequate (i.e, condition (b) in Definition 2 holds). Monotonicity guarantees that the final prefix generated by ERV will never contain a cut-off event that ERV failed to identify as such.

3.4 Size of the Finite Prefix

Up to now, we have been mainly concerned with the case when t_R is reachable. Next, we discuss some results related to the size of the prefix generated in case it is not. For this, we need an additional property of the heuristic function h: we say that h is *safely pruning* iff $h(C) = \infty$ implies that there is no configuration

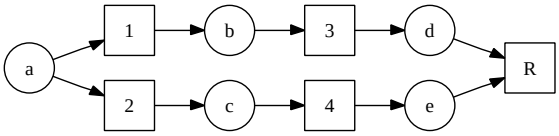


Fig. 1. Example net with an unreachable goal transition (R).

$C' \supseteq C$ with $t_R \in C'$, i.e. $h^*(C) = \infty$. Pruning safety is a weaker property than (general) admissibility, as it pertains only to a subset of configurations (the dead-end configurations from which the goal is unreachable). Most heuristic functions safisfy it; in particular so do all the specific heuristics we consider in this paper.

If h has this property, then the unfolding can be stopped as soon as the f-value of the best event retrieved from the queue is ∞, since this implies that t_R is unreachable. Note, however, that the prefix generated at this point is not necessarily complete: it may lack some markings that are reachable but irrelevant for the purpose of reaching t_R.

Let us illustrate the behavior of a safely pruning heuristic on the small example in Figure 1. Suppose initially only place a is marked: at this point, a heuristic such as $h^{\max}$ (see next section) estimates that the goal marking $\{d, e\}$ is reachable in 2 steps (the max length of the two paths). However, as soon as either transition 1 or 2 is taken, leading to a configuration in which either place b or c is marked, the $h^{\max}$ estimate becomes ∞, since there is then no way to reach one of the two goal places.

In general, the size of the prefix generated by directed unfolding is related to the informedness of the guiding heuristic. The following theorem, stated without proof due to lack of space, makes these relationships more precise. For a heuristic function h, let ERV(h) be the ERV algorithm directed with h, and let $\beta(h)$ be the prefix built by ERV(h) at termination.

Theorem 4. *Let $f^* = |C^*|$ be the cost of an optimal configuration C^*, and ∞ if no solution exists: (i) if h is admissible, then all events $e \in \beta(h)$ have f-value $\leq f^*$. (ii) If h_1 and h_2 are two monotone heuristics such that $h_1 \leq h_2$, and ties are broken systematically in the same fashion by ERV(h_1) and ERV(h_2), then $\beta(h_2) \subseteq \beta(h_1)$.*

Corollary 1. *If t_R is not reachable, the prefix $\beta(h)$ for any monotone heuristic h is no greater than that generated by the ERV algorithm with a breadth-first strategy.*

Proof. Follows from the theorem and the fact that ERV is equivalent to directed unfolding with $h \equiv 0$, which is monotone. □

While the result guarantees only that directed unfolding with a monotone heuristic can not do worse than the breadth-first strategy in the unreachable case, in the following we demonstrate experimentally that in practice it often does significantly better, generating much smaller prefixes than blind ERV. What determines the size of the generated prefix is mainly the *pruning power* of the heuristic, i.e., its ability to assign an infinite cost estimate to configurations from

which the goal marking is unreachable. The heuristics we consider in this paper are all equivalent in this respect, but others, such as e.g. pattern database (PDB) heuristics [15], have much greater pruning power.

4 Heuristics

A common approach to constructing heuristic functions, both admissible and inadmissible, is to define a *relaxation* of the search problem, such that the relaxed search problem can be solved, or at least approximated, efficiently, and use the cost of the relaxed solution as an estimate of the cost of the solution to the real problem, i.e. as the heuristic value [16]. The problem of extending a configuration C of the unfolding into one that includes the target transition t_R is equivalent to the problem of reaching t_R starting from $\text{Mark}(C)$: this is the problem that we relax to obtain an estimate of the distance to reach t_R from C.

The heuristics we have experimented with are derived from two different relaxations, both developed in the area of AI planning. The first relaxation is to consider each place in the preset of a transition independently of the others. For a transition t to fire, each place in ${}^\bullet t$ must be marked: thus, the estimated distance from a given marking M to a marking where t can fire is $d(M, {}^\bullet t) = \max_{p \in {}^\bullet t} d(M, \{p\})$, where $d(M, \{p\})$ denotes the estimated distance from M to any marking that includes $\{p\}$. For a place p to be marked – if it isn't marked already – at least one transition in ${}^\bullet p$ must fire: thus, $d(M, \{p\}) = 1 + \min_{t \in {}^\bullet p} d(M, {}^\bullet t)$. Combining the two facts we obtain

$$d(M, M') = \begin{cases} 0 & \text{if } M' \subseteq M \\ 1 + \min_{t \in {}^\bullet p} d(M, {}^\bullet t) & \text{if } M' = \{p\} \\ \max_{p \in M'} d(M, \{p\}) & \text{otherwise} \end{cases} \quad (1)$$

for the estimated distance from a marking M to M'. The solution to equation (1) can be computed in polynomial time using dynamic programming. We obtain a heuristic function, called $h^{\max}$, by $h^{\max}(C) = d(\text{Mark}(C), {}^\bullet t_R)$. This estimate is never greater than the actual distance, so the $h^{\max}$ heuristic is admissible.

In many cases, however, it is too weak to effectively guide the unfolding. Admissible heuristics in general tend to be conservative (since they need to ensure that the distance to the goal is not overestimated) and therefore less discriminating between different states. Inadmissible heuristics, on the other hand, have a greater freedom in assigning values and are therefore often more informative, in the sense that the relative values of different states is a stronger indicator of how "promising" the states are. An inadmissible, but often more informative, version of the $h^{\max}$ heuristic, called h^{sum}, can be obtained by substituting $\sum_{p \in M'} d(M, \{p\})$ for the last clause of equation (1).

The second relaxation is known as the *delete relaxation*. In Petri net terms, the simplifying assumption made in this relaxation is a transition only requires the presence of a token in each place in its preset, but does not consume those tokens when fired (put another way, all arcs leading into a transition are assumed to be read-arcs). This implies that a place once marked will never be unmarked, and therefore that any reachable marking is reachable by a "short" transition

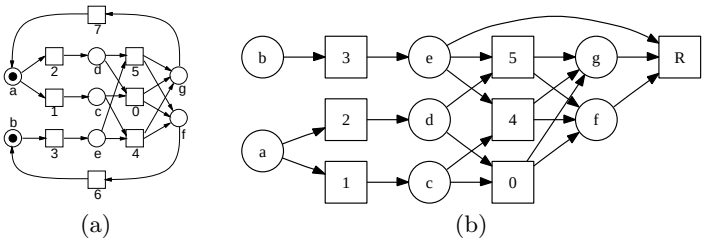


Fig. 2. (a) Example PT-net with marking and (b) corresponding relaxed plan graph.

sequence. Every marking that is reachable in the original net is a subset of a marking that is reachable in the relaxed problem. The delete-relaxed problem has the property that a solution – if one exists – can be found in polynomial time. The procedure for doing this constructs a so called "relaxed plan graph", which is essentially a complete prefix of the unfolding of the relaxed problem. Because of the delete relaxation, the construction of the relaxed plan graph is much simpler than the unfolding of a Petri net, and the resulting graph is conflict-free[6] and of bounded size (each transition appears at most once in it). Once the graph has been constructed, a solution (configuration leading to t_R) is extracted; in case there are multiple transitions marking a place, one is chosen arbitrarily. The size of the solution to the relaxed problem gives a heuristic function, called h^{FF} (after the planning system FF [9] which was the first to use it). Figure 2 shows an example of a marked net and the corresponding relaxed plan graph: a minimal solution is the sequence $3, 5, R$; other solutions include, e.g., $1, 3, 4, R$ and $1, 2, 0, 3, R$. The FF heuristic satisfies the conditions required to preserve the completeness of the unfolding (in Theorem 1), but, because an arbitrary solution is extracted from the relaxed plan graph, it is not admissible. The heuristic defined by the size of the *minimal* solution to the delete-relaxed problem, known as h^+, is admissible, but solving the relaxed problem optimally is NP-hard [17].

5 Experimental Results

We extended Mole to use the $\prec_f$ ordering with the $h^{\max}$, h^{sum}, and h^{FF} heuristics. In our experiments below we compare the resulting directed versions of Mole with the original (breadth-first) version, and demonstrate that the former can solve much larger instances than were previously within the reach of the unfolding technique. We found that the additional tie-breaking comparisons used by Mole to make the order strict were slowing down all versions (including the original): though they do – sometimes – reduce the size of the prefix, the computational overhead quickly consumes any advantage. (As an example, on the unsolvable random problems considered below, the total reduction in size amounted to less than 1%, while the increase in runtime was around 20%.) We

[6] Technically, delete relaxation can destroy the 1-safeness of the net. However, the exact number of tokens in a place does not matter, but only whether the place is marked or not, so in the construction of the relaxed plan graph, two transitions marking the same place are not considered a conflict.

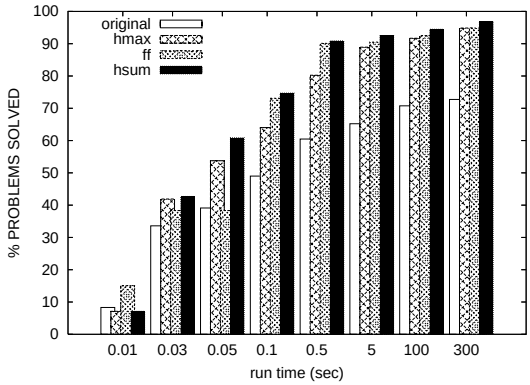


Fig. 3. Results for DARTES Instances

therefore disabled them in all experiments[7]. Experiments were conducted on a Pentium M 1.7GHz with a 2Gb memory limit. The nets used in the experiments can be found at `http://rsise.anu.edu.au/~thiebaux/benchmarks/petri`.

5.1 Petri Net Benchmarks

From the developers of MOLE we obtained a set of standard Petri net benchmarks representative of Corbett's examples [18]. Only one of them, DARTES, which models the communication skeleton of an Ada program, turned out to be a challenge for MOLE; for other benchmarks in the set, it generates even the complete finite prefix in a matter of seconds.

Figure 3 compares the performance of the original version of MOLE to the versions directed by each of the heuristics. For each of the 253 DARTES transitions, we recorded the time taken by each version to decide this transition's reachability. The graph shows the percentage of problems solved within increasing time limits, ranging from 0.01 to 300 sec. The original breadth-first version of MOLE is systematically outperformed by all of the directed versions. Overall, the original version is able to decide 185 of the 253 problem instances (73%), whereas the version directed by h^{sum} solves 245 of them (97%). The instances solved by each of the directed versions is a strict superset of those solved by the original. Unsurprisingly, all the solved problems were positive decisions (the transitions were reachable). Lengths of shortest solutions to DARTES instances reach up to over 90, and breadth-first could not solve any of the instances that had a shortest solution of length above 60.

5.2 Random Problems

To further investigate the scalability of directed unfolding, we implemented our own generator of random Petri nets. Conceptually, the generator creates a set of component automata, and connects them in an acyclic dependency network. The

[7] This means the original version of MOLE in our experiments implements McMillan's ordering [1].

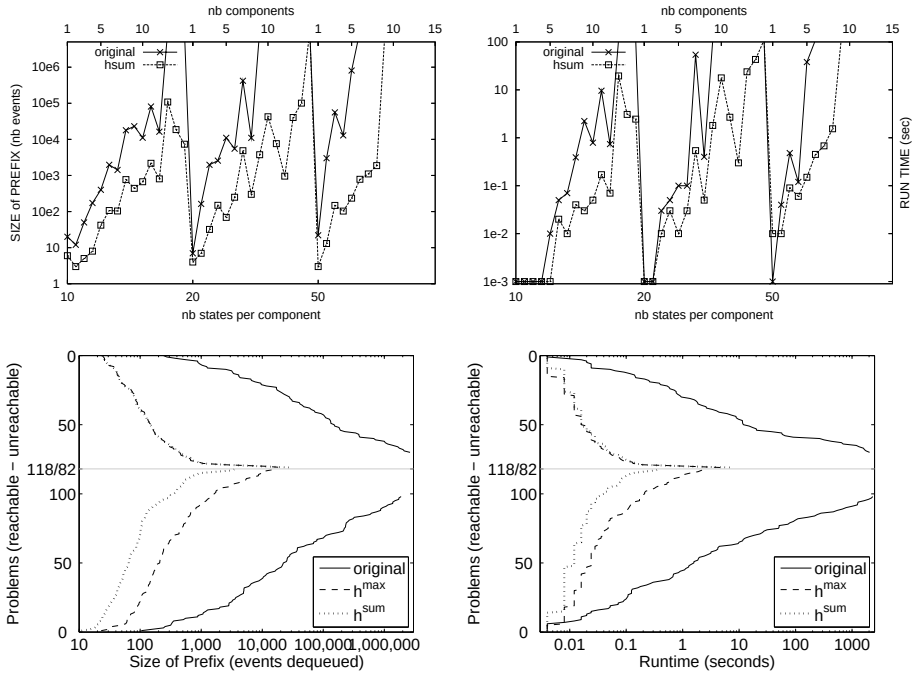


Fig. 4. Results for Random PT-nets

transition graph of each component automaton is a sparse, but strongly connected, random digraph. Synchronisations between pairs of component automata are such that only one (the dependent) automaton changes state, but can only do so when the other component automaton is in a particular state. Synchronisations are chosen randomly, constrained by the acyclic dependency graph. Target states for the various automata are chosen independently at random. The construction ensures that every choice of target states is reachable. We generated random problems featuring $1 \ldots 15$ component automata of 10, 20, and 50 states each. The resulting Petri nets range from 10 places and 30 transitions to 750 places and over 4000 transitions.

Results are shown in the top row of Figure 4. The left-hand graph shows the number of events pulled out of the queue. The right-hand graph shows the run-time. To avoid cluttering the graphs, we show only the performance of the worst and best strategy, namely the original one, and h^{sum}. Evidently, directed unfolding can solve much larger problems than blind unfolding. For the largest instances we considered, the gap reached over 2 orders of magnitude in speed and 3 in size. The original version could merely solve the easier half of the problems, while directed unfolding only failed on 6 of the largest instances (with 50 states per component).

In these problems, optimal firing sequences reach lengths of several hundreds events. On instances which we were able to solve optimally using $h^{\max}$, h^{FF} produced solutions within a couple transitions of the optimal. Over all problems, solutions obtained with h^{sum} were a bit longer than those obtained with h^{FF}.

With only a small modification, viz. changing the transition graph of each component automaton into a (directed) tree-like structure instead of a strongly connected graph, the random generator can also produce problems in which the goal marking has a fair chance of being unreachable. To explore the effect of directing on the unfolding in this case, we generated 200 such instances (each with 10 components of 10 states per component), of which 118 turned out to be reachable and 82 unreachable, respectively. The bottom row of Figure 4 shows the results, in the form of distribution curves (prefix size on the left and run-time on the right; note that scales are logarithmic). The lower curve is for solvable problems, while the upper, "inverse" curve, is for problems where the goal marking is not reachable. Thus, the point on the horizontal axis where the two curves meet on the vertical is where, for the hardest problem instance, the reachability question has been answered.

As expected, h^{sum} solves instances where the goal marking is reachable faster than $h^{\max}$, which is in turn much faster than blind unfolding. However, also in those instances where the goal marking is not reachable, the prefix generated by directed unfolding is significantly smaller than that generated by the original algorithm. In this case, results of using the two heuristics are nearly indistinguishable. This is due to the fact that, as mentioned earlier, their pruning power (ability to detect dead end configurations) is the same.

5.3 Planning Benchmarks

To assess the performance of directed unfolding on a wider range of problems with realistic structure, we also considered benchmarks from the two last editions of the International Planning Competition (IPC-4 and IPC-5). These benchmarks are described in PDDL (the Planning Domain Definition Language), which we translate into 1-safe PT-nets as explained in [4].

In the top of Figure 5, we present results for the first 26 IPC-4 instances of AIRPORT, a ground air-traffic control problem. Both the optimal and non-optimal AIRPORT planning problem are known to be PSPACE-complete [19]. The corresponding Petri nets range from 78 places and 18 transitions (instance 1) to 4611 places and 1711 transitions (instance 26). Optimal solution lengths range from 8 to over 200. As before, the left-hand graph shows the number of events pulled out of the queue, and the right-hand graph shows the run-time. To avoid cluttering the graphs, we do not show the performance of h^{sum}. Its curves are comprised between those for h^{FF} and $h^{\max}$. For small instances, the relatively small gain (1 order of magnitude fewer nodes) in unfolding size does not compensate for the overhead incurred in computing the heuristic function. However, for larger instances, directed unfolding reduces both size and run time by over 2 orders of magnitude. The original version of MOLE is unable to solve 6 of the instances within a 600 second time limit. These instances describe ground traffic control problems over the topology of half of Munich airport. $h^{\max}$ fails to solve the two larger instances, but h^{FF} solves them easily.

In the bottom of Figure 5 we present results for OPENSTACKS, a production scheduling problem. Optimal OPENSTACKS is NP-complete [20], while the problem becomes polynomial if optimality is not required. We consider instances

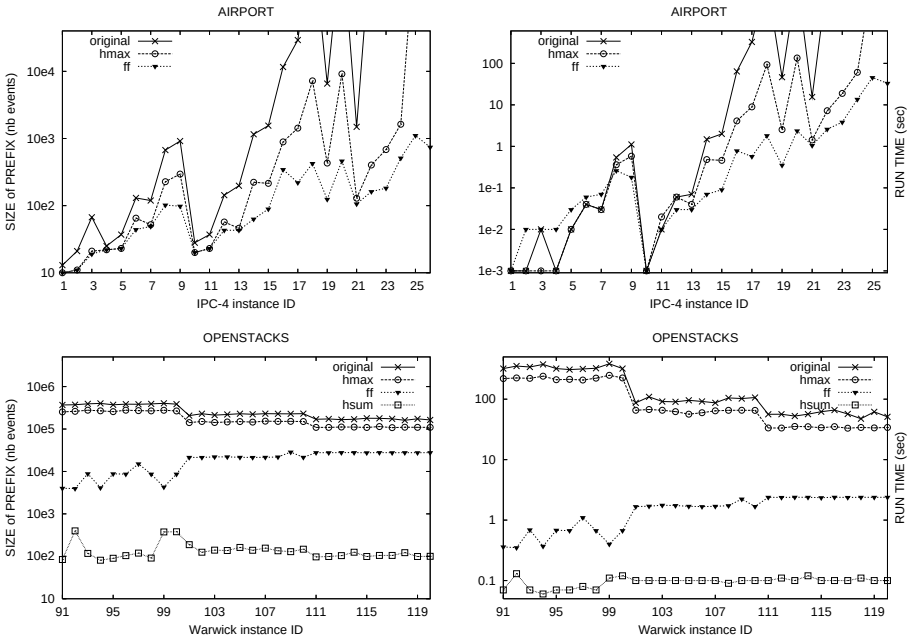


Fig. 5. Results for Planning Problems AIRPORT (top) and OPENSTACKS (bottom)

Warwick 91-120 which feature 10 products, 10 orders and an increasing ratio of 3 to 5 of products per order. The IPC-5 "propositional" version of OPENSTACKS disables concurrency. In contrast, while still retaining the IPC-5 optimality criterion, we use the natural encoding of OPENSTACKS which allows several products to be produced in parallel. The corresponding Petri nets all have 65 places and 222 transitions, but differ in their initial markings. The optimal solution length varies between 35 and 40 operations. In OPENSTACKS, the gap between directed and breadth-first unfolding is spectacular. The h^{sum} heuristic consistently spend around 0.1 sec solving the problem, that is over 3 orders of magnitude less than the breadth-first version. h^{FF}'s run time ranges from 0.3 sec (instance 91) to 2.8 sec (instance 120). This shows that directed unfolding, which unlike breadth-first search is not confined to optimal solutions, is able to exploit the fact that non-optimal OPENSTACKS is an easy problem.

6 Conclusion, Related and Future Work

We have described directed unfolding, which incorporates heuristic search straight into an on-the-fly reachability analysis technique specific to Petri nets. We proved that the ERV unfolding algorithm can benefit from using heuristic search strategies, whilst preserving finiteness and completeness of the generated prefix. Such strategies are effective for on-the-fly reachability analysis, as they significantly reduce the prefix explored to find a desired marking or to prove that none exists. We demonstrated that suitable heuristic functions, which estimate the distance between

configurations, can be automatically extracted from the original net. Both admissible and non-admissible heuristics can be used, with the former offering optimality guarantees. Experimental results illustrate that directed unfolding provides a significant performance improvement over the original breadth-first implementation of ERV featured in MOLE.

Edelkamp and Jabbar [21] recently introduced a method for directed model-checking Petri nets. It operates by translating the deadlock detection problem into a metric planning problem, solved using off-the-shelf heuristic search planning methods. These methods, however, do not exploit concurrency in the powerful way that unfolding does. In contrast, our approach combines the best of heuristic search and Petri net reachability analysis. The runtimes we obtain in our experiments with planning benchmarks are often competitive with those of the best performing planners. Importantly, this is achieved with an unfolding algorithm which does not handle read-arcs. The treatment of read arcs is essential to improve the performance of directed unfolding applied to planning, and is a high priority item on our future work agenda.

In this paper we have measured the cost of a configuration C by its cardinality, i.e. $g(C) = |C|$. Or similarly, $g(C) = \sum_{e \in C} c(e)$ with $c(e) = 1\ \forall e \in E$. These results extend to transitions having arbitrary non-negative cost values, i.e. $c : E \to \mathbb{R}$. Consequently, using any admissible heuristic strategy, we can find the minimum cost firing sequence leading to t_R. As in the cardinality case, the algorithm is still correct using non-admissible heuristics, but does not guarantee optimality. The use of unfolding for solving optimisation problems involving cost, probability and time, is a focus of our current research.

We also plan to use heuristic strategies to guide the unfolding of higher level Petri nets, such as coloured nets [22]. Our motivation, again arising from our work in the area of planning, is that our translation from PDDL to PT-nets is sometimes the bottleneck of our planning via unfolding approach [4]. Well developed tools such as PUNF[8] could be adapted for experiments in this area.

Note to the reader At the time of submission of our camera-ready copy, it has been pointed out to us that the completeness proof in [5] on which ours relies may lack significant detail which may or may not compromise its use for our purpose. We therefore invite the community to re-examine whether semi-adequacy confers completeness. Even if it does not, we note that completeness is guaranteed with monotone heuristics, as those can be recast as adequate orderings for which ERV is known to be complete [23]. Moreover, even if semi-adequacy does not guarantee completeness, this does not necessarily imply that using inadmissible heuristics compromises completeness. Finally, even if completeness turned out to be lost in the inadmissible case, our experimental results show that such incomplete methods might in many cases be more useful than complete ones.

Acknowledgements Many thanks to the anonymous reviewers and to workshop chair Victor Khomenko whose comments helped to significantly improve this paper. Thanks to Jussi Rintanen and John Slaney for interesting discussions,

[8] `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/tools.html`

to Stefan Schwoon for his help with Mole, and to Lang White for suggesting we explore the connections between planning and unfolding-based reachability analysis. The authors thank NICTA and DSTO for their support via the DPOLP project. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the ARC.

References

1. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: CAV. (1992) 164–177
2. Esparza, J.: Model checking using net unfoldings. Science of Compututer Programming **23**(2-3) (1994) 151–195
3. Benveniste, A., Fabre, E., Jard, C., Haar, S.: Diagnosis of asynchronous discrete event systems, a net unfolding approach. IEEE Transactions on Automatic Control **48**(5) (2003) 714–727
4. Hickmott, S., Rintanen, J., Thiébaux, S., White, L.: Planning via Petri net unfolding. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07). (2007) 1904–1911
5. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Formal Methods in System Design **20**(3) (2002) 285–310
6. Esparza, J., Schröter, C.: Unfolding based algorithms for the reachability problem. Fundamentia Informatica **46** (2001) 1–17
7. Esparza, J., Kanade, P., Schwoon, S.: A negative result on depth first unfolding. Software Tools for Technology Transfer (To appear)
8. Bonet, B., Geffner, H.: Planning as heuristic search: New results. In: Proceedings of the Ninth International Conference on Automated Planning and Scheduling (ICAPS/ECP-99). (1999) 360–372
9. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
10. McDermott, D.V.: Using regression-match graphs to control search in planning. Artificial Intelligence **109**(1-2) (1999) 111–159
11. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
12. Chatain, T., Khomenko, V.: A note on the well-foundedness of adequate orders used for truncating unfoldings. Technical Report 998, Newcastle University, School of Computing Science (2007)
13. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Proceedings of the Thirteenth Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS-93). (1993) 326–337 LNCS 761.
14. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit model checking with hsf-spin. In: Proceedings of the Eighth International SPIN Workshop. (2001) 57–79
15. Culberson, J., Schaeffer, J.: Searching with pattern databases. In: Canadian Conference on AI. Volume 1081 of LNCS., Springer (1996) 402–416
16. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley (1984)
17. Bylander, T.: The computational complexity of propositional strips planning. Artificial Intelligence **69**(1-2) (1994) 165–204
18. Corbett, J.C.: Evaluating deadlock detection methods for concurrent software. IEEE Transactions on Software Engineering **22**(3) (1996)
19. Helmert, M.: New complexity results for classical planning benchmarks. In: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006). (2006) 52–61

20. Linhares, A., Yanasse, H.: Connection between cutting-pattern sequencing, VLSI design and flexible machines. Computers & Operations Research **29** (2002) 1759 – 1772
21. Edelkamp, S., Jabbar, S.: Action planning for directed model checking of petri nets. Electronic Notes Theoretical Computer Science **149**(2) (2006) 3–18
22. Khomenko, V., Koutny, M.: Branching processes of high-level petri nets. In: Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-03). (2003) 458–472
23. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of petri net unfoldings. Acta Informatica **40**(2) (2003) 95–118

Modular construction of finite and complete prefixes

A. Madalinski and E. Fabre

IRISA/INRIA
Campus de Beaulieu
35042 Rennes cedex, France

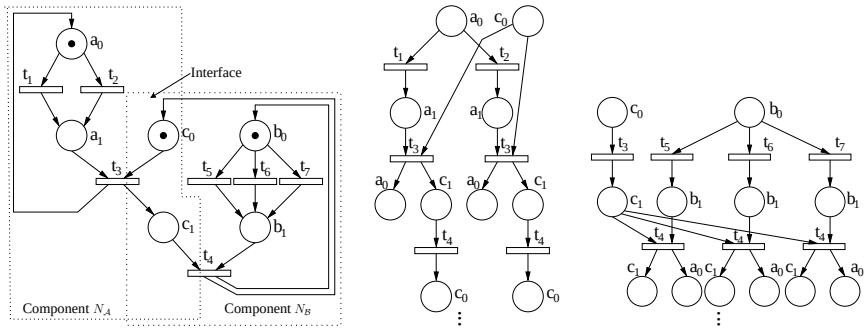
Abstract. This paper presents the modular construction of finite and complete prefixes applied to distributed systems, which are modelled by Petri nets in form of synchronous products of automata. A distributed system is described as a collection of components interacting through interfaces. They exhibit factorisation properties, viz. the unfolding of a distributed system factorises as the product of unfoldings of its components. This gives a compact representation and makes it possible to analyse the system by parts.
The construction of modular prefixes is based on deriving 'summaries' of components w.r.t. their interfaces whilst passing them on the interaction structure of system via interfaces. Prefixes of components are then derived locally by taking into account the summaries received on their interfaces.

1 Introduction

Petri nets (PNs) are a widely used model for analysing concurrent and distributed systems. Their unfoldings [3, 16] are convenient to represent concurrency due to their partial order semantics, where executions are considered as partially ordered sets of events rather than sequences. The finite and complete prefix [5, 12, 15] of a PN unfolding is a compact representation containing all information about the original system, and therefore, it is applied to model checking [11, 14].

The representation of a system can be further 'compressed' by an unfolding factorisation [1, 6, 7] when applied to distributed systems. In a distributed system two neighbouring components interact through an interface, a shared sub-system. The unfolding of a distributed system decomposes as the product of its components, and the collection of the local unfoldings may be more compact than the unfolding of the global system. This is illustrated in Figure 1. The system in Figure 1(a) consists of two components, $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$, interacting through an interface. There exist $m = 2$ possibilities to produce t_4 in $Unf_{N_{\mathcal{A}}}$ (Figure 1(b)) and $n = 3$ possibilities to produce t_4 in $Unf_{N_{\mathcal{B}}}$ (Figure 1(c)), hence $m \cdot n = 6$ different combinations in the unfolding of the entire system.

Local unfoldings restricted by the behaviour of the global system can be derived in a modular manner by exchanging information between interacting components; no global information is needed. The factorised representation is



(a) N (b) Unf_{N_A} (c) Unf_{N_B}

Fig. 1. Distributed system and the unfoldings of its components

the basis of modular or distributed processing. This framework has been applied to modular failure diagnosis [9] with application to telecommunication networks. In order to be a practical tool for modular model checking finite and complete prefixes of local unfoldings have to be derived. Precisely this is the objective of this paper.

A technique to construct a complete prefix of a synchronous product of labeled transition systems is presented in [4]. This technique uses the product structure of the model to simplify the construction, however, the construction is not modular. A modular construction of complete prefixes adapted to systems composed of Petri nets has been presented in [2]. However, the system used is restricted to non-reentrance synchronisations, i.e. one transition of a component can synchronise only with one transition in another component. In addition, the derivation of local prefixes uses global informations. In this paper only local information are used, which are passed in form of a summary net via interfaces between interacting components. A summary net describes an interface restricted by its component, and at the same time it carries minimal information about its component.

The paper is organised as follows. Section 2 and 3 gives the basic theoretical background concerning PNs and their unfoldings, and distributed systems and their factorisation properties. In Section 4 the concept of the modular derivation of finite a complete prefix of factorised unfoldings is presented. The conclusion and future work are discussed in Section 5.

2 Nets and unfoldings

In this section basic definitions concerning Petri nets and net unfoldings are presented. These are mainly adapted from [5, 12].

2.1 Petri Nets

A net is a quadruple $N = (P, T, \rightarrow, M^0)$ such that P and T are disjoint sets of *places* and *transitions*, respectively, $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, and $P^0 : P \rightarrow \mathbb{N} = \{0, 1, 2, ...\}$ is a multiset on P representing the *initial marking* of the net. For a node $x \in P \cup T$, its *pre-set* ${}^\bullet x$ is defined by ${}^\bullet x = \{y \mid (y, x) \in \rightarrow\}$ and its *post-set* $x^\bullet$ is defined by $x^\bullet = \{y \mid (x, y) \in \rightarrow\}$.

A net N is called k-bounded if, for every reachable marking M and every place $p \in P$, $M(p) \leq k$. In this paper only *safe* nets are considered, i.e. 1-bounded nets.

2.2 Branching processes

Two nodes of a net N, y and y', are in *structural conflict*, denoted by $y \# y'$, if there exist distinct transitions $t, t' \in T$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, and (t, y) and (t', y') are in the reflexive transitive closure of the flow relation $\rightarrow$, denoted by $\preceq$.

An *occurrence net* is a net $ON = (C, E, \rightarrow, C^0)$, where C is a set of *conditions* (places), E is the set of *events* (transitions) and $C^0 = c \in C : {}^\bullet c = \emptyset$ is the set of initial conditions satisfying the following: for every $c \in C, \mid {}^\bullet c \mid \leq 1$; for every $y \in C \cup E$, $(y \# y)$ and there are finitely many y' such that $y' \prec y$, where $\prec$ denotes the *causal relation*, the transitive closure of $\rightarrow$. Two nodes are *concurrent*, denoted $y \parallel y'$, if neither $y \# y'$ nor $y \preceq y'$ nor $y' \preceq y$.

A homomorphism from ON to N[1], also called a *folding*, is a mapping $h : C \cup E \rightarrow P \cup T$ such that $h(C) \subseteq P$ and $h(E) \subseteq T$ (conditions are mapped to places and events to transitions); for all $e \in E$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet h(e)$ and similarly for $e^\bullet$ and $h(e)^\bullet$ (transitions environments are preserved); the restriction of h to C^0 is a bijection between C^0 and P^0 (minimal conditions correspond to the initial marking); and for all $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$ then $e_1 = e_2$ (BP does not duplicate the transitions in N). A *branching process* of a net system N is a pair $BP = (ON, h)$ such that ON is an occurrence net and h is a homomorphism from ON to N. The (virtual) initial event is denoted by $\perp$, which has empty preset, the post-set C^0 and no label; it is assumed to exist without drawing it in figures.

A branching process BP' of N is a *prefix* of a branching process BP, denoted by $BP' \sqsubseteq BP$, if ON' is a causally closed sub-net of ON containing all initial conditions and such that: $\forall e \in E, e \in ON'$ implies $e^\bullet \subseteq ON'$ and h' is the restriction of h to $C' \cup E'$. For each net system N there exists a unique (up to isomorphism) maximal (w.r.t $\sqsubseteq$) branching process Unf_N (or short Unf), called the *unfolding* of N.

Configurations and cuts A *configuration* of a branching process BP is a finite set of events $\kappa \subseteq E$ such that for all $e, f \in \kappa$, $(e \# f)$ and, for every $e \in \kappa$, $f \prec e$ implies $f \in \kappa$; in addition it is required that $\perp \in \kappa$. For every event

[1] Note that a general definition of net morphism is presented in Section 3.2.

$e \in E$, the configuration $[e] \stackrel{\text{df}}{=} \{f | f \preceq e\}$ is called the *basic configuration*[2] of e , and $\langle e \rangle \stackrel{\text{df}}{=} [e] \setminus \{e\}$ denotes the set of *causal predecessors*. The set of all finite (basic) configurations of a branching process BP is denoted by $\boldsymbol{\kappa}_{fin}^{BP}(\boldsymbol{\kappa}_{bas}^{BP})$, and the superscription BP is dropped in the case $BP = Unf_N$.

A *co-set* is a set of condition C' such that for all distinct $c, c' \in C'$,$c \parallel c'$, and a *cut* is a maximal co-set for the set inclusion. Let κ be a configuration then $Cut(\kappa) \stackrel{\text{df}}{=} \left(C^0 \cup \kappa^\bullet\right) \setminus {}^\bullet\kappa$ is a cut; furthermore, the multiset of places $h\left(Cut(\kappa)\right)$ is a reachable marking of N, which is denoted by $Mark(\kappa)$. A marking M of N is *represented* in BP if there is a configuration κ of BP such that $M = Mark(\kappa)$. Every marking represented in BP is reached in N, and every reachable marking of N is represented in Unf_N.

There exist different methods of truncating PN unfoldings. The differences are related to the kind of information about the unfolding which are to be preserved in the prefix, as well as to the choice between using either only basic or all finite configurations. The former can improve the running time of an algorithm, and the latter can result in a smaller prefix.

Cutting context An abstract parametric model has been introduced in [11, 12] to cope with different variants of the technique for truncating unfoldings. It uses parameters which determine the information intended to be preserved in the complete prefix (in the standard case, this is the set of reachable markings) and specify the circumstances under which an event can be designated as a cut-off event.

Definition 1. *A* cutting context *is a triple* $\Theta = \left(\approx, \lhd, \{\boldsymbol{\kappa}_e\}_{e \in E}\right)$*, where:*

1. $\approx$ *is an equivalence relation on* $\boldsymbol{\kappa}_{fin}$*.*
2. $\lhd$*, called an* adequate *order, is a strict well-founded partial order on* $\boldsymbol{\kappa}_{fin}$ *refining* $\subset$*, i.e.* $\kappa' \subset \kappa''$ *implies* $\kappa' \lhd \kappa''$*.*
3. $\approx$ *and* $\lhd$ *are* preserved by finite extensions*, i.e. for every pair of configurations* $\kappa' \approx \kappa''$*, and for every suffix* E' *of* κ'*, there exists a finite suffix* E'' *of* κ'' *such that*
   (a) $\kappa'' \oplus E'' \approx \kappa' \oplus E'$*, and*
   (b) *if* $\kappa'' \lhd \kappa'$ *then* $\kappa'' \oplus E'' \lhd \kappa' \oplus E'$*.*
4. $\{\boldsymbol{\kappa}_e\}_{e \in E}$ *is a family of subsets of* $\boldsymbol{\kappa}_{fin}$*.*

The adequate order specifies which configurations are preserved in the complete prefix; all $\lhd$-minimal configurations in each equivalent class of $\approx$ are preserved. The last parameter is needed to specify the set of configurations used later to decide whether an event can be designed as a cut-off event. The cutting context $\Theta_{ERV} = \left\{\approx_{mar}, \lhd_{tot}, \{\boldsymbol{\kappa}_e = \boldsymbol{\kappa}_{bas}\}_{e \in E}\right\}$ corresponds to the framework in [5], where $\approx_{mar}$ is the equivalence relation on reachable marking of N, i.e. $\kappa' \approx_{mar} \kappa''$ iff $Mark(\kappa') = Mark(\kappa'')$, and $\lhd_{tot}$ is a total adequate order.

[2] The term local configuration is used by several authors, however, here the term 'local' is ambiguous.

Canonical prefixes The static cut-off events are defined independent of an unfolding algorithm, where feasible events are events whose causal predecessors are not static cut-off events and thus are included in the prefix determined by those cut-off events.

Definition 2. *The set of* feasible *events, denoted by* $fsble^{\Theta}$*, and the set of* static cut-off *events, denoted by* cut^{Θ}*, are two sets of events of Unf defined inductively, in the following way:*

1. *An event* e *is a feasible event if* $\langle e\rangle \cap cut^{\Theta} = \emptyset$.
2. *An event* e *is a static cut-off event if it is feasible, and there is a configuration* $\kappa \in \boldsymbol{\kappa}_e$ *such that* $\kappa \subseteq fsble^{\Theta} \setminus cut^{\Theta}$, $\kappa \approx [e]$, *and* $\kappa \lhd [e]$. *In the sequel, every* κ *satisfying these conditions will be called a* corresponding *configuration of* e.

The notion of canonical prefix arises quite naturally, after observing that $fsble_N^{\Theta}$ is a downward-closed set of events.

Definition 3. *The branching process* $Pref_N^{\Theta}$ *induced by the set of events* $fsble_N^{\Theta}$ *is called the* canonical prefix *of Unf.*

Note that $Pref_N^{\Theta}$ is uniquely determined by the cutting context Θ. Several fundamental properties of $Pref_N^{\Theta}$ have been proven in [12]. In particular, $Pref_N^{\Theta}$ is always complete w.r.t. cut_N^{Θ}, and it is finite if $\approx$ has finitely many equivalence classes and $\boldsymbol{\kappa}_e \supseteq \boldsymbol{\kappa}_{bas}$.

3 Distributed systems

Distributed systems are modeled by Petri nets in form of Multi-clock nets. They are described as a collection of components interacting via interfaces. Their factorisation property makes it possible to process them in a modular manner. This section is based on works in [6, 7, 10].

3.1 Multi-clock nets

Multi-clock net (MCN) is a tuple $N = \left(P, T, \rightarrow, P^0, \nu\right)$, where $\left(P, T, \rightarrow, P^0\right)$ is an ordinary safe Petri net, $\nu : P \rightarrow P^0$ is a partition on places, and $\forall t \in T, \nu$ is injective on ${}^\bullet t$ and on $t^\bullet$, and $\nu\left({}^\bullet t\right) = \nu\left(t^\bullet\right)$. In an MCN, the number of tokens remains constant, and in any reachable marking $P' \subseteq P$ one has $\nu : P' \rightarrow P^0$ is bijective. In other words, $\forall p \in P$, the restriction $N_{|\overline{p}}$ of N to places $\overline{p} = \nu^{-1}\left(\nu\left(p\right)\right)$ is a sequential machine, i.e. an ordinary automaton. Therefore, an MCN can be regarded as synchronous product of automata. An MCN with three classes is depicted in Figure 2.

A multi-clocked labeled net $N = \left(P, T, \rightarrow, P^0, \nu, \lambda, \Lambda\right)$ is an MCN extended with a labelling function $\lambda : T \rightarrow \Lambda$ on transitions and a label set Λ.

Given a safe Petri net it is always possible to obtain an MCN with essentially the same behaviour by, for example, introducing complementary places. Thus, the limitation of MCNs is not a strong assumption. MCNs are convenient to represent components and used for projections on components.

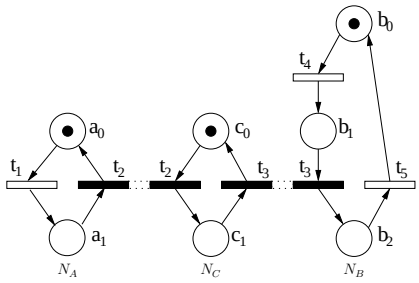


Fig. 2. Decomposition of a multi-clock net

3.2 Compound systems

The synchronous product of labeled nets can be applied to combined nets to a larger system. Given two nets N_A and N_C the synchronous product, denoted by $N_A \times_N N_C$, synchronises (glues) transitions with identical label, preserves transitions with private labels and forms a disjoint union of places. This is illustrated in Figure 2 where N_A and N_C synchronise on transitions labeled t_2 and N_B and N_C synchronise on transitions labeled t_3. Note that transitions names are used as labels sets.

In this paper distributed systems are modeled by a set of interconnected components interacting through interfaces. Formally, such a composition is expressed as a special case of a pullback [8]. A pullback generalises the product to the case of components sharing some places and transitions. Let first recall the notion of morphism between nets [17]. A morphism $\phi : N_A \to N_B$ between $N_x = \left(P_x, T_x, \to_x, P_x^0, \nu_x\right)$, where $x \in \{A, B\}$, is a pair $\left(\phi^P, \phi^T\right)$ with ϕ^P a relation on places and ϕ^T a relation on transitions[3]. The initial marking is preserved by ϕ as follows: $P_B^0 = \phi\left(P_A^0\right)$ and $\forall p_B \in P_B^0, \exists! p_A \in P_A^0 : p_A \phi p_B$. If ϕ is defined on $p_A \in P_A$, then it is also defined on both ${}^\bullet p_A$ and $p_A^\bullet$; ϕ preserves the environment of each transition: $t_B = \phi\left(t_A\right)$ implies that restrictions $\phi : {}^\bullet t_A \to {}^\bullet t_B$ and $\phi : t_A^\bullet \to t_B^\bullet$ are both bijective. In the case of an MCN it is required that ϕ preserves partitions of places: $\forall\left(p_A, p_B\right) \in P_A \times P_B, p_A \phi p_B \Rightarrow \nu_A\left(p_A\right) \phi \nu_B\left(p_B\right)$.

Definition 4. *Let $N_\mathcal{A}, N_\mathcal{B}$ and N_C be labeled nets, and $\phi_x : N_x \to N_C$, $x \in \{\mathcal{A}, \mathcal{B}\}$, be two net morphisms such that ϕ_x are partial functions on places (no place duplications). The* pullback *of this triple, denoted by $N = N_\mathcal{A} \times_N^{N_C} N_\mathcal{B}$, is defined on places by*

$$\begin{aligned} P = & \left\{\left(p_\mathcal{A}, \star\right) : p_\mathcal{A} \in P_\mathcal{A}, p_\mathcal{A} \notin Dom\left(\phi_\mathcal{A}\right)\right\} \\ & \cup \left\{\left(\star, p_\mathcal{B}\right) : p_\mathcal{B} \in P_\mathcal{B}, p_\mathcal{B} \notin Dom\left(\phi_\mathcal{B}\right)\right\} \\ & \cup \left\{\left(p_\mathcal{A}, p_\mathcal{B}\right) \in P_\mathcal{A} \times P_\mathcal{B} : \phi_\mathcal{A}\left(p_\mathcal{A}\right) = \phi_\mathcal{B}\left(p_\mathcal{B}\right)\right\} \end{aligned} \tag{1}$$

and on transitions by

[3] For simplicity, ϕ is written instead of ϕ^Por ϕ^T.

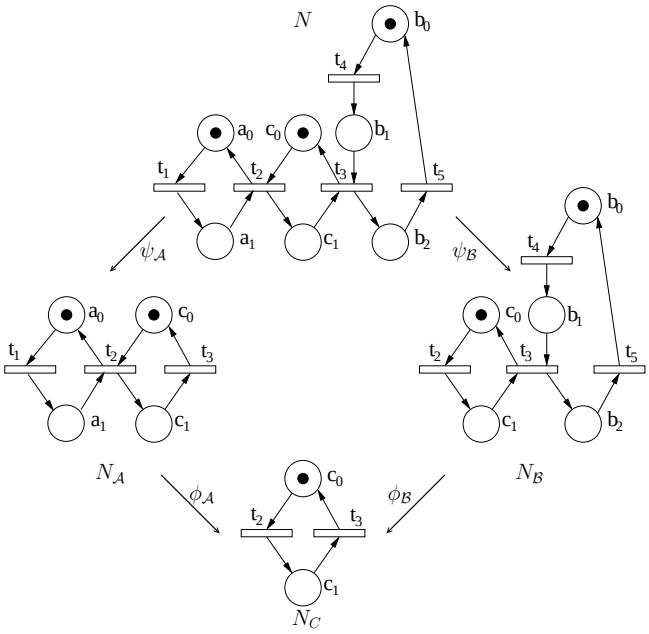


Fig. 3. The pullback $N = N_{\mathcal{A}} \times_N^{N_C} N_{\mathcal{B}}$

$$\begin{aligned} T = &\{(t_{\mathcal{A}}, \star) : t_{\mathcal{A}} \in T_{\mathcal{A}}, t_{\mathcal{A}} \notin Dom\left(\phi_{\mathcal{A}}\right), \lambda_{\mathcal{A}}\left(t_{\mathcal{A}}\right) \in \Lambda_{\mathcal{A}} \setminus \Lambda_{\mathcal{B}}\} \\ &\cup \{(\star, t_{\mathcal{B}}) : t_{\mathcal{B}} \in T_{\mathcal{B}}, t_{\mathcal{B}} \notin Dom\left(\phi_{\mathcal{B}}\right), \lambda_{\mathcal{B}}\left(t_{\mathcal{B}}\right) \in \Lambda_{\mathcal{B}} \setminus \Lambda_{\mathcal{A}}\} \\ &\cup \{(t_{\mathcal{A}}, t_{\mathcal{B}}) \in T_{\mathcal{A}} \times T_{\mathcal{B}} : t_x \notin Dom\left(\phi_x\right), \lambda_{\mathcal{A}}\left(t_{\mathcal{A}}\right) = \lambda_{\mathcal{B}}\left(t_{\mathcal{B}}\right)\} \\ &\cup \{(t_{\mathcal{A}}, t_{\mathcal{B}}) \in T_{\mathcal{A}} \times T_{\mathcal{B}} : \phi_{\mathcal{A}}\left(t_{\mathcal{A}}\right) = \phi_{\mathcal{B}}\left(t_{\mathcal{B}}\right)\} \end{aligned} \tag{2}$$

The flow relation follows accordingly as well as the definition of initial places.

Moreover, $\psi_x : N \rightarrow N_x$ denotes the canonical morphism that maps elements of N to the corresponding elements in N_x.

A special case of a pullback is applied where N_C is an *interface* between $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$ iff transitions of N_x with the same label ($\Lambda_{\mathcal{A}} \cap \Lambda_{\mathcal{B}}$) are all in the definition domain of ϕ_x. Thus, there is no interaction between $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$ 'outside' the domains of ϕ_x, which is characterised by the disappearance of the third line in (2). Therefore, the two components interact only through a shared sub-system, the interface.

A labeled multi-clock net N is said to be a *distributed system* if it can be expressed as a pullback of several components where the intermediary nets are interfaces. The global interaction structure of a distributed system can be represented as a graph, where an edge is drawn between two components if they have a common interface.

For simplicity in this paper the case with two components interacting via an interface is used. Furthermore, it is assumed that the interface is an automaton,

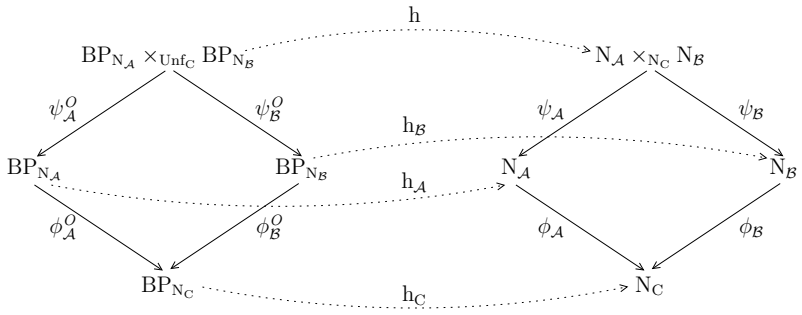


Fig. 4. Commutative diagram of the pullback

not a general MCN. This assumption avoids dealing with some technical aspects of projections of unfoldings. Figure 3 depicts the distributed system $N = N_{\mathcal{A}} \times_N^{N_C} N_{\mathcal{B}}$ with associated morphisms $\phi_x : N_x \rightarrow N_C$, where $N_{\mathcal{A}} = N_A \times_N N_C$ and $N_{\mathcal{B}} = N_B \times_N N_C$.

3.3 Factorisation of unfoldings

It was shown in [6] that the factorised form of a net system yields a factorised form of the unfolding of such a system. Given $N = N_{\mathcal{A}} \times_N^{N_C} N_{\mathcal{B}}$ one obtains

$$Unf_N = Unf_{\mathcal{A}} \times_O^{Unf_C} Unf_{\mathcal{B}}, \tag{3}$$

where Unf_N, $Unf_{\mathcal{A}}$ and $Unf_{\mathcal{B}}$ are the unfoldings of N, $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$, and $\times_O^{Unf_C}$ denotes the pullback on branching processes. Thus, the unfolding of a global system can be expressed as a pullback of unfoldings of components (as illustrated in Figure 4).

Let $BP_{N_x} = \left(C_x, E_x, \rightarrow_x, C_x^0, h_x\right)$ be branching processes of N_x, where $x = \{\mathcal{A}, \mathcal{B}\}$. The *pullback*

$$BP_{N_{\mathcal{A}}} \times_O^{Unf_C} BP_{N_{\mathcal{B}}} = Unf_{BP_{N_{\mathcal{A}}} \times_N^{Unf_C} BP_{N_{\mathcal{B}}}}, \tag{4}$$

which is yields a branching process of $N_{\mathcal{A}} \times_N^{N_C} N_{\mathcal{B}}$. There exist a recursive procedure to compute the pullback of BP_{N_x}, where a pullback is performed under the constraint that the resulting net remains a branching process.

Given $Unf = Unf_{\mathcal{A}} \times_O^{Unf_C} Unf_{\mathcal{B}}$. The *restriction* of Unf to nodes labelled by elements of N_C is denoted by $Unf_{|N_C}$. The *projection* of Unf on behaviours of N_C is defined as

$$\Pi_C\left(Unf\right) = \phi_{\mathcal{A}} \circ \psi_{\mathcal{A}}\left(Unf\right) = \phi_{\mathcal{B}} \circ \psi_{\mathcal{B}}\left(Unf\right). \tag{5}$$

It is obtained by performing $Unf_{|N_C}$ and by trimming isomorphic configurations. The latter is necessary since the restriction may produce several isomorphic

copies of configurations. The trimming consists of eliminating the redundant copies in order to get a branching process. It is obtained by a recursive procedure starting at minimal conditions and 'merging' events which have the same pre-set and the same label; when merging events also their post-sets are merged.

The restriction of *Unf* to a subset of nodes may erase causality or conflict relations between these nodes, which could appear as unduly concurrent in the restriction. A projection $\Pi_C(Unf)$ is said to be *non misleading* if every configuration κ' in $\Pi_C(Unf)$ is the image of a configuration of κ in *Unf*, and causality relations on events on κ' are not lost. It was shown in [6] that the projections on automata are always non misleading.

The main result in [6] is that minimal factors of *Unf* can be obtained in a modular manner without computing *Unf* itself:

$$\Pi_{\mathcal{A}}(Unf) = Unf_{\mathcal{A}} \times_O^{Unf_C} \Pi_C(Unf_{\mathcal{B}}) \tag{6}$$

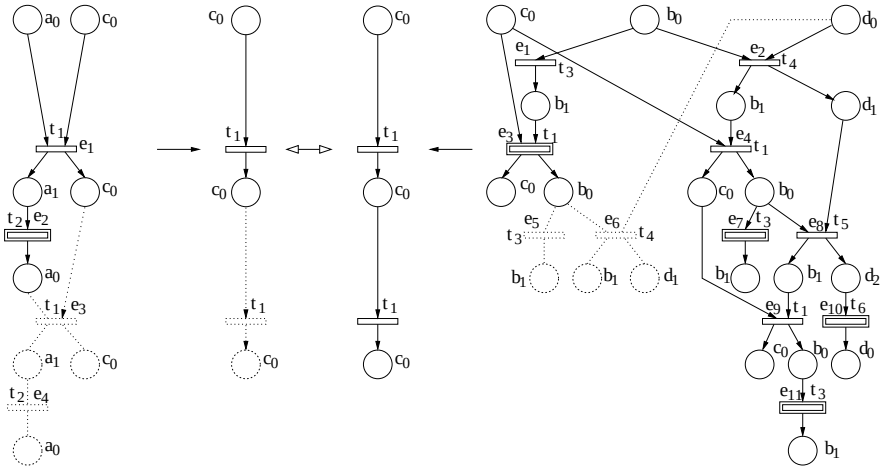
(and the symmetrically for $\Pi_{\mathcal{B}}(Unf)$). Minimal factors of more complex distributed system living on a tree can be obtained by a messaging passing algorithm, which runs on the interaction structure and progressively updates information on interfaces. The modular computation of minimal factors holds on branching processes provided projections are non misleading.[4] In this paper the derivation of minimal factors of finite and complete prefixes is undertaken. This is not a trivial task, and therefore the distributed system is limited to the case of two components, which interact only through a common interface. However, the presented approach can be generalised to more complex tree-shaped systems.

4 Modular complete prefixes

The construction of modular complete prefixes of $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$ requires the exchange of some behaviours from one component to the other one and vice versa. These behaviours include the interface restricted to a component and the local marking reached by a component on its interface. However, it is not known how much information the other component needs; this means in terms of prefixes how far should a component be unfolded in order to offer the other component with the necessary informations.

For example, Figure 5 depicts $Pref_{N_{\mathcal{A}}}$ and $Pref_{N_{\mathcal{B}}}$, two locally complete prefixes of components with a simple interface only consisting of the transition t_1 and the place c_0. $Pref_{N_{\mathcal{A}}}$ need one occurrence of the interface transition t_1. In contrast, $Pref_{N_{\mathcal{B}}}$ needs two occurrence of t_1 to be complete. Thus, $Pref_{N_{\mathcal{A}}}$ has to be extended by one occurrence of t_1; this new behaviour on the interface has to be propagated to $Pref_{N_{\mathcal{B}}}$. In addition, one still needs to find the cut-off points by considering global markings, and this might require the extension of the prefixes of these components until global truncation points are reached. This might

[4] This limitation was overcome in [7] by introducing augmented branching processes to capture extra causality and conflict relations. In [6, 7] approximate minimal factors are obtained for systems not living on a tree



(a) $Pref_{N_\mathcal{A}}$ (b) interaction (c) $Pref_{N_\mathcal{B}}$

Fig. 5. Illustrating required interaction via the interface between components

result in many exchanges. To avoid this situation the behaviour of a component on its interface is captured in form of a summary net, which is obtained from an extended canonical prefix by 'folding' the part corresponding to that interface.

4.1 Extended canonical prefixes

The extended canonical prefix of a component $N_\mathcal{A}$ or $N_\mathcal{B}$ is built with regard to its interface. Such a prefix captures the behaviour of that interface in relation to its component. It is obtained by restricting the cutting context, in particular the set of configurations which are used for the cut-off criterion.

Definition 5. *Let $N_\mathcal{A}$ be a component and N_C an interface of $N_\mathcal{A}$. Let $\boldsymbol{\kappa}_{bas}^{N_C}$ be the set of all basic configurations of the unfolding of $N_\mathcal{A}$ restricted to events corresponding to N_C. Then, w.r.t. the interface N_C, the cutting context $\Theta_{N_C} = \left(\approx_{mar}, \lhd_{tot}, \{\boldsymbol{\kappa}_e\}_{e \in E}\right)$ is defined with $\forall e \in E_\mathcal{A}$,*

$$\boldsymbol{\kappa}_e = \begin{cases} \{\boldsymbol{\kappa}_p \in \boldsymbol{\kappa}_{bas} \mid \Pi_C\left([e] \triangle \boldsymbol{\kappa}_p\right) = \emptyset\} & \text{if } \Pi_C\left(e\right) = \emptyset, \\ \boldsymbol{\kappa}_{bas}^{N_C} & \text{otherwise} \end{cases},$$

where $\triangle$ is the symmetric set difference.

The restriction of the cutting context Θ_{N_C} in $N_\mathcal{A}$ means that an interface event (an event corresponding to the interface N_C) can be designated as a cut-off event if its corresponding event is also an interface event. Whereas, the corresponding event e' of a private cut-off event e (i.e. an event which do not correspond to the interface N_C) has to be chosen such that there are no interface events in $[e] \triangle [e']$.

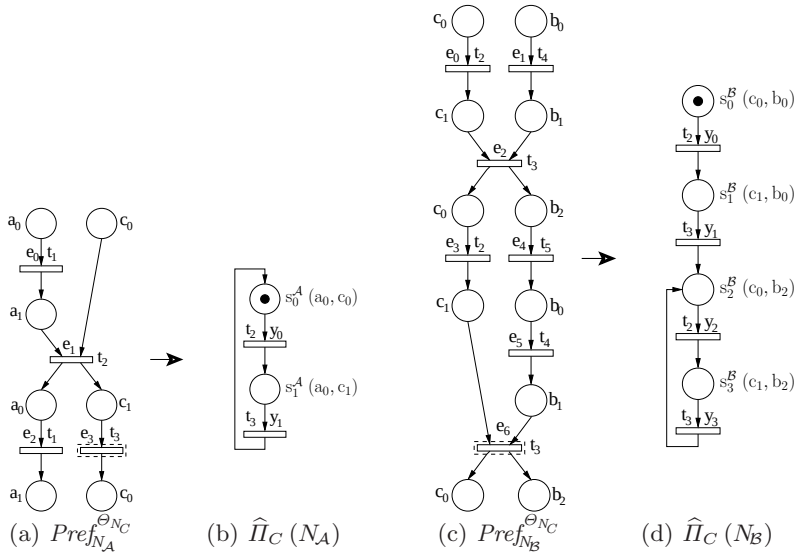


Fig. 6. Extended prefixes and their corresponding summary nets

Definition 6. *The branching process* $Pref_{N_\mathcal{A}}^{\Theta_{N_C}}$ *induced by the set of events* $fsble_{N_\mathcal{A}}^{\Theta_{N_C}}$ *is called the* extended canonical prefix *of* $N_\mathcal{A}$ *w.r.t. its interface* N_C*.*

To distinguish static cut-off events between canonical prefixes and extended ones the cut-offs of the former are drawn as double boxes and for the latter the outer box will be drawn as dashed line. Figure 6(a) and (c) shows the extended canonical prefixes of the components $N_\mathcal{A}$ and $N_\mathcal{B}$, in Figure 3. $Pref_{N_\mathcal{A}}^{\Theta_{N_C}}$ coincides with the canonical prefix since the only cut-off event and its corresponding event are interface events.

However, this is not the case for $Pref_{N_\mathcal{B}}^{\Theta_{N_C}}$ in Figure 6(c). The extended prefix is larger than the standard prefix, which would be obtained by setting e_4 as a cut-off event since $[\bot] \approx_{mar} [e_4]$, $[\bot] \lhd_{tot} [e_4]$ and $[e_4] \in \boldsymbol{\kappa}_{bas}$. However, e_4 does not corresponds to an extended cut-off event since it is a private event and $\Pi_C\,([e_4] \triangle [\bot]) = \{t_2, t_3\}$. This applies also to the event e_5. The event e_6 is an extended cut-off since it and its corresponding event e_2 are interface events.

An extended canonical prefix is complete since it is a canonical prefix (see Proposition 2.9 in [11]). It has to be shown that it is finite.

Proposition 1. $Pref_{N_\mathcal{A}}^{\Theta_{N_C}}$ *is finite.*

Proof. By Proposition 2.10 in [11] it is enough to show that each infinite $\prec$-chain in $Unf_{N_\mathcal{A}}$ can be cut. Two cases are considered. If there are infinitely many interface events, i.e. events which correspond to N_C, in the chain then the chain can be cut since the number of markings is finite and thus some marking is a final marking of several interface events.

Otherwise, there is only a finite number of interface events in the chain. Since the chain is infinite it contains an infinite tail of only private (non-interface) events. Since the number of markings is finite some marking is a final marking of several private events in the tail and thus the chain can be cut.

4.2 Summary nets

The summary net of a component captures the behaviour of a component w.r.t. its interface, and it is derived from its extended canonical prefix.

Definition 7. *Let* $Pref_{N_\mathcal{A}}^{\Theta_{N_C}}$ *be the extended canonical prefix of* $N_\mathcal{A}$ *w.r.t. its interface* N_C *with the set of static cut-off events* $cut_{N_\mathcal{A}}^{\Theta_{N_C}}$. *Let* $L = (S, Y, s_0, \rightarrow)$ *denote the automaton* $loop((Pref_{N_\mathcal{A}}^{\Theta_{N_C}})_{|N_C})$ *with loop defined as*

$$\forall e \in cut_{N_\mathcal{A}}^{\Theta_{N_C}} : \Pi_C(e) \neq \emptyset \text{ merge } \Pi_C(e^\bullet) \text{ and } \Pi_C(e'^\bullet),$$

where e' *is the corresponding event of* e *(i.e.* $e' \lhd e$ *and* $e' \approx_{mar} e$*). The initial state* s_0 *corresponds to the minimal conditions of* $\Pi_C\left(Pref_{N_\mathcal{A}}^{\Theta_{N_C}}\right)$. *In the sequel,* $loop((Pref_{N_\mathcal{A}}^{\Theta_{N_C}})_{|N_C})$ *is called the* summary net *of* $N_\mathcal{A}$ *w.r.t. its interface* N_C *and is denoted by* $\widehat{\Pi}_C(N_\mathcal{A})$.

The loop function merges interface conditions corresponding to the direct successors of interface cut-off events and their corresponding events. The cutting context Θ_{N_C} is designed in such a way that interface cut-off events have a corresponding interface event, and thus are in the restriction of the interface. Note that due to the restriction of interfaces to automata the summary nets are also automata.

The transitions of a summary net are associated with interface events of the corresponding extended prefix and the states are associated with markings reached by those interface transitions in that component. Figure 6 depicts the summary nets of the components in Figure 3 together with their corresponding extended prefixes. The net $\widehat{\Pi}_C(N_\mathcal{A})$ in Figure 6(b) coincides with the interface N_C. This is not the case with $\widehat{\Pi}_C(N_\mathcal{B})$ in Figure 6(d). There are two states labeled by either c_0 or c_1, however, they are associated with different marking in the component which are given in brackets next to the states. It can be seen that summary nets carry minimal information about components since their states are linked with the markings reached by interface transitions.

Observe that the summary net is obtained by looping the restriction instead of the projection of the extended prefix to the behaviours of the interface N_C (no trimming is applied). The trimming would merge states in the summary net, which correspond to conditions in the resulting projection. However, these states might be associated with different markings of a component, and thus cannot me merged. One could apply a trimming at the level of summary nets to isomorphic configurations having equivalent merged markings.

The states of the summary net are associated with unique markings since the extended prefix, from which the summary net is derived, uses the total adequate order. This means in terms of interface events that having two interface events with equal final marking implies that one is associated with a cut-off event, and thus they are merge by the looping.

Let $f_{\mathcal{B}}$ be a mapping from $\widehat{\Pi}_C(N_{\mathcal{B}})$ to N_C with $f_{\mathcal{B}} = \phi_{\mathcal{B}} \circ h_{\mathcal{B}}$, where $\phi_{\mathcal{B}} : N_{\mathcal{B}} \rightarrow N_C$ and $h_{\mathcal{B}} : ON_{\mathcal{B}} \rightarrow N_{\mathcal{B}}$. Then, the pullback $N_{\mathcal{A}} \times_N^{N_C} \widehat{\Pi}_C(N_{\mathcal{B}})$ is defined with $\phi_{\mathcal{A}} : N_{\mathcal{A}} \rightarrow N_C$ and $f_{\mathcal{B}} : \widehat{\Pi}_C(N_B) \rightarrow N_C$.

Due to the nature of the summary net the following relation holds.

Proposition 2. $\Pi_C(Unf_{\mathcal{A}}) = Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})}$.

Proof. From the Definition 7 it follows that $Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})} \times_O^{Unf_C} Unf_{\mathcal{A}} = Unf_{\mathcal{A}}$, which in turn is equal to $Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})} \times_O Unf_A$. Furthermore, $\Pi_C\left(Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})} \times_O Unf_A\right) = Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})}$ since $\widehat{\Pi}_C(N_{\mathcal{A}})$ is constructed from the extended complete prefix of $N_{\mathcal{A}}$ it contains all the behaviour induced by $N_{\mathcal{A}}$ on N_C, and hence, no behaviour is lost or added by Unf_A to $Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})}$. Thus, it can be concluded that $\Pi_C(Unf_{\mathcal{A}}) = Unf_{\widehat{\Pi}_C(N_{\mathcal{A}})}$.

4.3 Modular canonical prefix

Given a distributed system $N = N_{\mathcal{A}} \times_N^{N_C} N_{\mathcal{B}}$ the *modular* complete prefix of a component $N_{\mathcal{A}}$ is obtained by

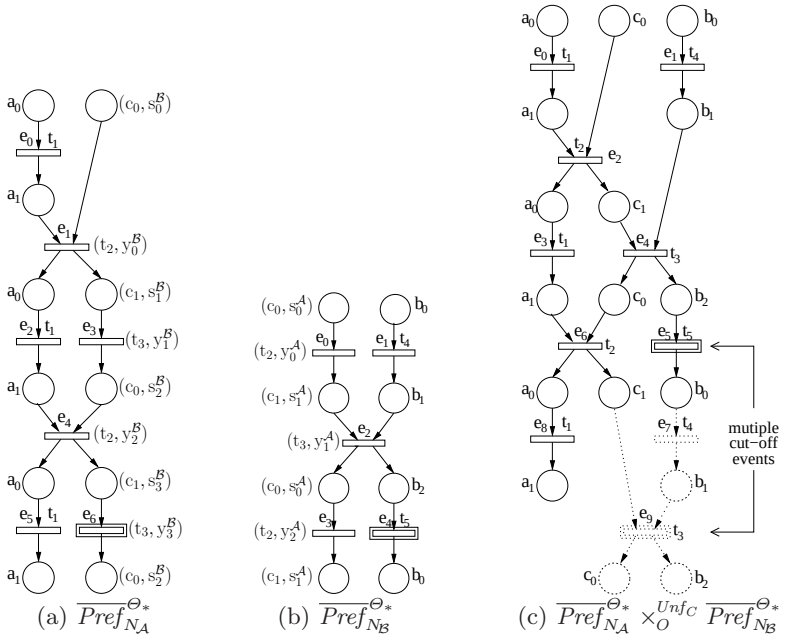
$$N_{\mathcal{A}\leftarrow C} = N_{\mathcal{A}} \times_N^{N_C} \widehat{\Pi}_C(N_{\mathcal{B}}), \tag{7}$$

$$\overline{Pref}_{N_{\mathcal{A}}}^{\Theta_*} = Pref_{N_{\mathcal{A}\leftarrow C}}^{\Theta_*}, \tag{8}$$

where $\Theta_* = \left\{\approx_{mar}, \subset, \{\boldsymbol{\kappa}_e = \boldsymbol{\kappa}_{bas}\}_{e\in E}\right\}$. The symmetric relation holds for $\overline{Pref}_{N_{\mathcal{B}}}^{\Theta_*}$. In this set-up the basic adequate order allows the comparison of basic configurations locally and globally in an analogous manner. This is possible due to the information provided on interfaces by the summary nets. Note that there exist a mapping $\overline{h_{\mathcal{A}}} = \psi_{\mathcal{A}} \circ h_{\mathcal{A}}$ from $\overline{Pref}_{N_{\mathcal{A}}}^{\Theta_*}$ to $N_{\mathcal{A}}$, where $\psi_{\mathcal{A}} : N_{\mathcal{A}\leftarrow C} \rightarrow N_{\mathcal{A}}$ and $h_{\mathcal{A}} : ON_{\mathcal{A}} \rightarrow N_{\mathcal{A}\leftarrow C}$.

Conservative cut-off criteria are the consequence of building such modular complete prefixes. These are caused by restricted possibilities to obtain cut-off events and by independent cut-off events in components.

Due to the nature of the contraction some possible cut-off candidates are not visible locally. A component $N_{\mathcal{A}}$ receives information from $N_{\mathcal{B}}$ on its interface events. This allows a more restrictive set of markings than in the entire system, e.g. a marking reached locally in $Pref_{N_{\mathcal{B}}}$ is not visible in $Pref_{N_{\mathcal{A}}}$, only markings reached by interface transitions are seen by $Pref_{N_{\mathcal{A}}}$. The restriction is not too strong since only basic configurations are considered, and thus, this restriction



(a) $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$ (b) $\overline{Pref}^{\Theta_*}_{N_\mathcal{B}}$ (c) $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}} \times_O^{Unf_C} \overline{Pref}^{\Theta_*}_{N_\mathcal{B}}$

Fig. 7. Modular complete prefixes

only applies to interface events. In addition, non-visible markings in one component can be visible in others; only the combination of a visible and a not-visible marking in a component is not observed.

Independent truncation of unfoldings of components can result in multiple cut-off events in the entire system. Consequently, events in a prefix of a component might not be reachable globally due to an occurrence of a cut-off event in another prefix of a component. This situation is illustrated in Figure 7(c). There are two successive cut-off events e_5 and e_9, which correspond to cut-off events of $\overline{Pref}^{\Theta_*}_{N_\mathcal{B}}$ and $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$, respectively. However, the events e_7 and e_9 are not reachable in the entire system since e_5 is a cut-off event.

Proposition 3. $\Pi_\mathcal{A}\left(Pref^{\Theta_*}_{N_\mathcal{A} \times^{N_C}_N N_\mathcal{B}}\right) \sqsubseteq \overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$.

Proof. Observe that $\Pi_\mathcal{A}\left(Unf_{N_\mathcal{A} \times^{N_C}_N N_\mathcal{B}}\right) = Unf_{N_\mathcal{A} \times^{N_C}_N \widehat{\Pi}_C(N_\mathcal{B})}$ by Proposition 2 since $\widehat{\Pi}_C(N_\mathcal{B})$ contains the behaviour of the interface N_C restricted by $N_\mathcal{B}$. The resulting complete prefixes with the given cutting context Θ_* employ the same restriction on the interface N_C due to $N_\mathcal{B}$. The only difference is due to the conservative cut-off criteria, thus $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$ is, in general, larger than $\Pi_\mathcal{A}\left(Pref^{\Theta_*}_{N_\mathcal{A} \times^{N_C}_N N_\mathcal{B}}\right)$ since some global markings are invisible during the construction of $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$ and

there may occur cut-off events in the other component which would restrict the construction of $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$.

Accordingly, $Pref^{\Theta_*}_{N_\mathcal{A} \times^{N_C}_{N} N_\mathcal{B}} \sqsubseteq \overline{Pref}^{\Theta_*}_{N_\mathcal{A}} \times^{Unf_C}_{O} \overline{Pref}^{\Theta_*}_{N_\mathcal{B}}$ since in a global prefix cut-off events might be detected faster as in components due to invisibility of markings in components. In particular, an event e can be designated as a cut-off in the entire system with its corresponding event e', however, the corresponding configurations might not be seen at the level of the components. For example, e and e' correspond to local events of different components, which interact through an interface; in each component only one configuration corresponding to e and e', respectively, is visible, and thus, such a cut-off is not detected.

Proposition 4. $\overline{Pref}^{\Theta_*}_{N_\mathcal{A}}$ *is finite.*

The proof is similar to the one in Proposition 1.

It is shown in [13] that the above approach generalises to more complex tree-shaped systems having many components. In that case the modular construction procedure involves the generation of summary nets of components at the root and the leafs on the interaction structure. These nets are then propagated via their interfaces to the neighbour components and updated to the new interfaces of that component. The modular prefixes are generated when obtaining all summary nets on their interfaces.

Approximate modular complete prefixes can be obtained for distributed systems with general structures. These have interesting properties; if some given behaviour is forbidden by these approximate factors it is certainly forbidden in the true system, while the converse does not hold.

5 Conclusions

A modular construction of finite and complete prefixes of distributed systems is presented. This approach is based on exchanging summary nets of components through the interaction structure via interfaces. A summary net of a component is a compact representation of the behaviour of that component w.r.t one of its interfaces. Modular complete prefixes of components are built by considering the summary nets received on their interfaces. This is done purely locally without the use of global information.

The presented approach is applied to simple interfaces (i.e. automata). It is planned to extend this work to overcome this limitation. Furthermore, a prototype tool has been developed.

References

1. Paolo Baldan, Stefan Haar, and Barbara König. Distributed Unfolding of Petri Nets. In *FoSSaCS*, pages 126–141, 2006.

2. Jean-Michel Couvreur, Sébastien Grivet, and Denis Poitrenaud. Unfolding of products of symmetrical Petri nets. In José Manuel Colom and Maciej Koutny, editors, *Proceedings of the 22th International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 121–143. Springer Verlag, 2001.
3. Joost Engelfriet. Branching Processes of Petri Nets. *Acta Informatica*, 28:575–591, 1991. NewsletterInfo: 38, 40.
4. Javier Esparza and Stefan Romer. An unfolding algorithm for synchronous products of transition systems. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 2–20, 1999.
5. Javier Esparza, Stefan Römer, and Walter Vogler. An Improvement of McMillan's Unfolding Algorithm. In *Formal methods in systems design*, pages 285–310, 2002.
6. E. Fabre. Factorization of Unfoldings for Distributed Tile Systems, Part 1 : Reduced Interaction Case. Technical Report 1529, IRISA, April 2003.
7. E. Fabre. Factorization of Unfoldings for Distributed Tile Systems, Part 2 : General Case. Technical Report 1606, IRISA, May 2004.
8. E. Fabre. On the Construction of Pullbacks for Safe Petri Nets. In *"Applications and Theory of Petri Nets and other Models of Concurrency, ATPN'06, Turku, Finland"*, June 2006.
9. E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed Monitoring of Concurrent and Asynchronous Systems. *Journal of Discrete Event Systems, special issue*, pages 33–84, May 2005.
10. Eric Fabre. Distributed Diagnosis based on Trellis Processes. In *44th Conf. on Decision and Control (CDC), Seville, Spain*, December 2005.
11. V Khomenko. *Model Checking Based on Petri Net Unfolding Prefixes*. PhD thesis, University of Newcastle upon Tyne, 2002.
12. Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica, Volume 40, Number 2*, pages 95–118, October 2003.
13. A. Madalinski and E. Fabre. Modular construction of finite and complete prefixes. Technical report, IRISA, 2007. to appear.
14. K. L. McMillan. *Symbolic Model Checking: an approach to the state explosion problem*. PhD thesis, 1992.
15. K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. International Workshop on Computer Aided Verification*, pages 164–177, July 1992.
16. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part I. *Theor. Computer Science*, 13(1):85–108, January 1980.
17. Glynn Winskel. A New Definition of Morphism on Petri Nets. In *STACS '84: Proceedings of the Symposium of Theoretical Aspects of Computer Science*, pages 140–150, London, UK, 1984. Springer-Verlag.