# Verification of Petri Nets with Read Arcs

César Rodríguez and Stefan Schwoon

LSV (ENS Cachan & CNRS & INRIA), France

**Abstract.** Recent work studied the unfolding construction for contextual nets, i.e. nets with read arcs. Such unfoldings are more concise and can usually be constructed more efficiently than for Petri nets. However, concrete verification algorithms exploiting these advantages were lacking so far. We address this question and propose SAT-based verification algorithms for deadlock and reachability of contextual nets. Moreover, we study optimizations of the SAT encoding and report on experiments.

## 1 Introduction

Petri nets are a well-known model for concurrent systems. McMillan [17] introduced unfoldings as a tool for verifying properties of such nets. Roughly speaking, the unfolding of a net $N$ is an acyclic net bisimilar to $N$. McMillan showed that for bounded nets one can use a finite prefix $\mathcal{P}$ of the unfolding to check certain properties of $N$, e.g. reachability of markings or deadlock-freeness; McMillan himself proposed a deadlock-checking algorithm based on this idea.

The interest of unfoldings lies in the fact that, while $\mathcal{P}$ is in general larger than $N$, it is smaller than the full reachability graph. Moreover, deadlock or reachability checking are NP-complete for $\mathcal{P}$ but PSPACE-complete for $N$. Thus, the unfolding technique represents a time/space tradeoff for verifying Petri nets. This tradeoff is particularly attractive when testing multiple properties of the same net because $\mathcal{P}$ needs to be constructed only once.

The publication of [17] triggered a large body of research. To name a few items, the necessary size of $\mathcal{P}$ has been reduced [9], efficient tools for generating $\mathcal{P}$ have been implemented [16, 24], and unfoldings-based verification algorithms have been developed [7, 10, 11, 14, 18]. An extensive survey can be found in [8].

Recently, unfoldings of *contextual* nets (c-nets) have been studied, i.e. nets with *read arcs* that check for the presence of tokens without consuming them. Their unfoldings can be exponentially more compact than for Petri nets. It is thus natural to base verification on unfoldings of c-nets rather than Petri nets.

Previous work on c-net unfoldings has concentrated on their *construction*: [2] gave an abstract algorithm, and [1, 22] provided efficient construction methods. However, concrete verification algorithms making use of them are still missing. In this paper, we aim to close this gap. Our contributions are twofold: we investigate SAT-encodings of unfoldings, and we extend them to c-nets.

Concerning the first point, recall that given a finite complete prefix $\mathcal{P}$ of a bounded Petri net $N$, deciding deadlock-freeness, reachability, or coverability on $N$ is NP-complete. Thus, previous works consisted in reductions to different

NP-complete problems: McMillan [17] employed a branch-and-bound technique, Heljanko [11] a stable-models encoding, and Melzer and Römer [18] used mixed integer linear programming, later improved by Khomenko and Koutny [14, 15]. The technique used by Esparza and Schröter [10] is an ad-hoc algorithm based on additional information obtained while computing the unfolding.

The previous decade has seen the emergence of powerful SAT solvers. It is natural to profit from these advances and reduce to SAT instead; all the more so because unfoldings are 1-safe nets, so the marking of a place naturally translates to a boolean variable. Indeed, SAT solving has already been proposed for the similar problem of model-checking merged processes [13], and [8] gives an explicit SAT encoding for Petri net unfoldings. However, we are not aware of a publicly available tool that uses this idea. We examine the performance of the encoding and propose some optimizations.

Our principal contribution consists in extending the techniques for deadlock checking and reachability to unfoldings of c-nets. Thus, we intend to leverage their advantages w.r.t. ordinary unfoldings, i.e. faster construction and smaller size. It is worth noting that the smaller size of c-net unfoldings does not automatically translate to an easier SAT problem, for the following reasons: First, the presence of read arcs may cause so-called *cycles of asymmetric conflict*. Thus, a SAT encoding requires acyclicity constraints, which are not necessary for conventional unfoldings. Secondly, an event in a c-net unfolding can occur in multiple different execution contexts, called *histories*, and the constructions proposed in [1,2,22] require to annotate events with potentially many such histories. In contrast, every event in a Petri net unfolding has only one history. Some verification algorithms for Petri nets rely on this fact and do not easily adapt to c-nets. We propose solutions for both problems. Our encoding does not refer to the histories at all, and the effect of the acyclicty constraints can be palliated by several strategies. We add that the SAT-encoding for c-net unfoldings was already briefly sketched in [25], but without considering these problems.

To our knowledge, this is the first paper proposing practical verification algorithms using unfoldings of c-nets. These algorithms are provided as an add-on to the tool Cunf, which is freely available [20]. The tool is more efficient than previous approaches when applied to Petri net unfoldings, and even more efficient than that when used on c-net unfoldings.

The paper is structured as follows: In Section 2, we recall notation and previous results. In Section 3 we explain how unfoldings can be used to check for deadlock and reachability, and in Section 4, we discuss the reduction of the problem to SAT. We report on experiments in Section 5 and conclude in Section 6. A longer version of this paper is available at [21].

## 2 Basic notions

In this section, we establish our basic definitions and recall previous results. Due to space constraints, this section is quite concise (see [2, 22] for background).

### 2.1 Contextual nets

A *contextual net* (c-net) is a tuple $N = \langle P, T, F, C, m_0 \rangle$, where $P$ and $T$ are disjoint sets of *places* and *transitions*, $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, $C \subseteq P \times T$ is the *context relation* and $m_0 \colon P \to \mathbb{N}$ is the *initial marking*. A pair $(p, t) \in C$ is called *read arc*. A *Petri net* is a c-net without read arcs. $N$ is called *finite* if $P$ and $T$ are finite sets. Fig. 1 (a) depicts a c-net. Read arcs are drawn as undirected lines, here between $p$ and $C$.

For $x \in P \cup T$, let ${}^\bullet x := \{\, y \in P \cup T \mid (y, x) \in F \,\}$ the *preset* of $x$ and $x^\bullet := \{\, y \in P \cup T \mid (x, y) \in F \,\}$ the *postset* of $x$. The *context* of a place $p$ is defined as $\underline{p} := \{\, t \in T \mid (p, t) \in C \,\}$, and the context of a transition $t$ as $\underline{t} := \{\, p \in P \mid (p, t) \in C \,\}$. These notions extend to sets in the usual fashion.

A function $m \colon P \to \mathbb{N}$ is called *marking* of $N$. A transition $t$ is *enabled* at $m$ if $m(p) \geq 1$ for all all $p \in \underline{t} \cup {}^\bullet t$. Such, $t$ can *fire*, leading to marking $m'$, where $m'(p) = m(p) - |\{p\} \cap {}^\bullet t| + |\{p\} \cap t^\bullet|$ for all $p \in P$. We say that some marking $m$ is *reachable* if it can be obtained by a finite sequence of firings starting at $m_0$. A marking $m$ is *deadlocked* if it does not enable any transition.

$N$ is called *k-bounded* if $m(p) \leq k$ for all reachable $m$ and $p \in P$, and *safe* if it is 1-bounded. For safe nets, we treat markings as sets of places carrying tokens.

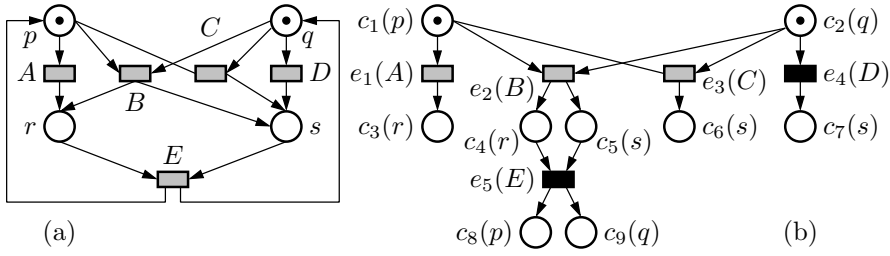


**Fig. 1.** (a) A safe c-net $N$; and (b) an unfolding prefix $\mathcal{P}$ for $N$.

### 2.2 Occurrence nets

Let $N = \langle P, T, F, C, m_0 \rangle$ be a c-net. For $t, t' \in T$, we write $t \lessdot t'$ if $t^\bullet \cap ({}^\bullet t' \cup \underline{t'}) \neq \emptyset$. We write $<$ for the transitive closure of $F \cup \lessdot$, and $\leq$ for the reflexive closure of $<$. For $x \in P \cup T$, we write $[x]$ for the set of *causes* of $x$, defined as $\{t \in T \mid t \leq x\}$. A set $X \subseteq T$ is *causally closed* if $[t] \subseteq X$ for all $t \in X$.

Two transitions $t, t'$ are in *symmetric conflict*, denoted $t \# t'$, iff ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, and in *asymmetric conflict*, written $t \nearrow t'$, iff (i) $t \lessdot t'$, or (ii) $\underline{t} \cap {}^\bullet t' \neq \emptyset$, or (iii) $t \neq t'$ and $t \# t'$. In case (ii) we also write $t \nearrow\!\!\!\!\nearrow t'$. For a set of events $X \subseteq T$, we write $\nearrow_X$ to denote the relation $\nearrow \cap\, (X \times X)$.

A c-net $O = \langle B, E, G, D, \widehat{m}_0 \rangle$ is called an *occurrence net* iff (i) $O$ is safe and for any $b \in B$, we have $|{}^\bullet b| \leq 1$; (ii) $<$ is a strict partial order for $O$; (iii) for all $e \in E$, $[e]$ is finite and $\nearrow_{[e]}$ acyclic; (iv) $\widehat{m_0} = \{\, b \in B \mid {}^\bullet b = \emptyset \,\}$.

Let $O$ be such an occurrence net. As per tradition, we call the elements of $B$ *conditions*, and those of $E$ *events*. A *configuration* of $O$ is a finite, causally closed set of events $\mathcal{C}$ such that $\nearrow_{\mathcal{C}}$ is acyclic. $\mathit{Conf}(O)$ denotes the set of all such configurations. For a configuration $\mathcal{C}$, let $\mathit{cut}(\mathcal{C}) := (\widehat{m_0} \cup \mathcal{C}^\bullet) \setminus {}^\bullet\mathcal{C}$.

A *prefix* of $O$ is a net $\mathcal{P} = \langle B', E', G', D', \widehat{m}_0 \rangle$ such that $E' \subseteq E$ is causally closed, $B' = \widehat{m}_0 \cup (E')^\bullet$, and $G', D'$ are the restrictions of $G, D$ to $(B' \cup E')$.

### 2.3 Unfoldings

Let $N = \langle P, T, F, C, m_0 \rangle$ be a bounded c-net. It is possible [2, 22] to produce an occurrence net $\mathcal{U}_N = \langle B, E, G, D, \widehat{m}_0 \rangle$, called the *unfolding* of $N$ and equipped with a mapping $f\colon (B \cup E) \to (P \cup T)$, that has the following properties:

- $f$ maps conditions to places and events to transitions. We extend $f$ to sets, multisets, and sequences in the usual way; $f$ applied to a marking of $\mathcal{U}_N$ (a set) will yield a marking of $N$ (a multiset).
- $\mathcal{U}_N$ is an acyclic version of $N$, i.e. the firing sequences and reachable markings of $\mathcal{U}_N$, modulo the mapping $f$, are exactly the same as in $N$.

In general, $\mathcal{U}_N$ is infinite, but one can generate a finite prefix $\mathcal{P}$ of it that is *marking-complete*, meaning that any marking $m$ is reachable in $N$ iff there exists a marking $\widehat{m}$ reachable in $\mathcal{P}$ with $f(\widehat{m}) = m$. Fig. 1 (b) depicts a marking-complete prefix of the c-net shown in Fig. 1 (a), where $f$ is given in parentheses.

## 3 Using unfoldings for verification

In this section, we illustrate why some existing verification approaches for Petri net unfoldings do not adapt well to c-net unfoldings. This justifies the choice of marking-completeness in Section 2.3 and is related to the notion of cutoff.

For Petri nets (i.e., without read arcs), existing algorithms such as [9, 17] produce a finite prefix $\mathcal{P}$ by truncating the unfolding at so-called *cutoff events*. Essentially, for a cutoff event $e$ there exists another event $e'$ in $\mathcal{P}$ such that $f(\mathit{cut}([e])) = f(\mathit{cut}([e']))$. Intuitively, $e$ does not contribute a new marking to the unfolding, and therefore $e$ and its successors can be omitted from $\mathcal{P}$.

Certain deadlock-checking algorithms for Petri nets depend on a stricter notion than marking-completeness, which we call *cutoff-completeness*, that also demands to include such cutoffs in $\mathcal{P}$. If $\mathcal{P}$ is cutoff-complete, then $N$ contains a deadlock iff $\mathcal{P}$ contains a cutoff-free configuration $\mathcal{C}$ such that $\mathit{cut}(\mathcal{C})$ is deadlocked in $\mathcal{P}$. This reduction is directly employed in [14, 18] and indirectly in [17].

Seeing as the algorithm in [14] performs very well, it would be tempting to adapt this reduction to c-nets. However, we provide an example showing that this reduction is problematic for c-nets. First, recall that the unfolding construction

for c-nets given in [1, 2, 22] lifts the notion of cutoff to event-history pairs. Here, a configuration $H$ is called *history* of an event $e$ if $e'(\nearrow_H)^* e$ for all $e' \in H$. In this case $\langle e, H\rangle$ is called *extended event*, and in analogy to Petri nets, some extended events will be marked as cutoffs when another extended event $\langle e', H'\rangle$ exists such that $f(cut(H)) = f(cut(H'))$. An event may have multiple histories, some of which are cutoffs while others are not.

The net shown in Fig. 1 (a) is free of deadlocks. An unfolding prefix $\mathcal{P}$ is shown in Fig. 1 (b), the mapping $f$ is given in parentheses. Event $e_1$ has two histories: $H_1 = \{e_1\}$ and $H_2 = \{e_3, e_1\}$. The unfolding algorithm will make $\langle e_1, H_2\rangle$ a cutoff but not $\langle e_1, H_1\rangle$; indeed $H_2$ leads to the same marking $\{r, s\}$ as $\langle e_2, \{e_2\}\rangle$[1]. An event is shown in black if all its histories are cutoffs.

The prefix in Fig. 1 (b) is marking-complete and also cutoff-complete, when the latter notion is lifted to enriched events. Under this assumption the above-given reduction of the deadlock-checking problem is still valid.

Consider the marking $m' = \{c_3, c_6\}$, which is deadlocked in $\mathcal{P}$. The configuration leading to $m'$ has a cutoff (namely, $\langle e_1, H_2\rangle$), so $m'$ cannot be interpreted as representing a deadlock of $N$ – indeed $f(m') = \{r, s\}$ enables transition $E$ in $N$. However, as this example demonstrates, *checking* whether a given configuration is cutoff-free requires to reason about histories and not just about events. This is undesirable because forbidding certain histories would result in a rather more complex SAT formula. We therefore use another solution that is completely event-based and requires only marking-completeness:

*Remark 1.* Let $N$ be a bounded c-net and $\mathcal{P}$ a marking-complete prefix for $N$. Then $N$ contains a deadlock iff $\mathcal{P}$ has a reachable marking $m'$ such that $f(m')$ is deadlocked. Moreover, $m$ is reachable in $N$ iff $\mathcal{P}$ has a reachable marking $m'$ such that $f(m') = m$.

In the following, we assume that every event in a marking-complete prefix has at least one non-cutoff history; the unfolding tool CUNF [20] can be instructed to remove the others at no extra cost.

## 4 SAT-encodings of c-nets

The SAT problem is as follows: given a formula $\phi$ of propositional logic, find whether there exists a satisfying assignment that makes $\phi$ true. SAT solving has taken a quantum leap during the last decade, and many efficient solvers for this problem exist. Here, we encode the deadlock-checking and reachability problem for c-nets in SAT, based on Remark 1. For Petri nets, such an encoding was given in [8]; we generalize it to c-nets and enrich it with optimizations. Notice that most constraints that we give translate directly into CNF.

For the rest of this section, let $N = \langle P, T, F, C, m_0\rangle$ be a finite safe c-net and $\mathcal{P} = \langle B, E, G, D, \widehat{m}_0\rangle$ a finite marking-complete prefix of $N$. We first construct

[1] It is not important to understand why the unfolding construction prefers to declare $\langle e_1, H_2\rangle$ a cutoff rather than $\langle e_2, \{e_2\}\rangle$, and our point is independent of this choice; what matters is that some events may have cutoff and non-cutoff histories.

a propositional formula $\phi_{\mathcal{P}}^{\text{dead}}$ that is unsatisfiable iff $N$ is deadlock-free. Section 4.4 explains the modifications needed to implement reachability checking, and Section 4.5 explains how the encoding can be generalized to bounded nets.

The formula $\phi_{\mathcal{P}}^{\text{dead}}$ is defined over variables $\mathsf{e}$ for $e \in E$ and $\mathsf{p}$ for $p \in P$ as:

$$\phi_{\mathcal{P}}^{\text{dead}} := \phi_{\mathcal{P}}^{\text{causal}} \wedge \phi_{\mathcal{P}}^{\text{sym}} \wedge \phi_{\mathcal{P}}^{\text{asym}} \wedge \phi_{\mathcal{P}}^{\text{mark}} \wedge \phi_{\mathcal{P}}^{\text{dis}}$$

The first three constraints enforce that any satisfying assignment represents a configuration $\mathcal{C}$, and $\phi_{\mathcal{P}}^{\text{mark}}$ defines the marking $m := f(cut(\mathcal{C}))$, which $\phi_{\mathcal{P}}^{\text{dis}}$ verifies to be deadlocked.

Recall that a configuration is a causally closed set of events free of loops in the $\nearrow$ relation. Subformulae $\phi_{\mathcal{P}}^{\text{causal}}$ and $\phi_{\mathcal{P}}^{\text{sym}}$ request $\mathcal{C}$ to be a causally closed set of events that has no pair of events in symmetric conflict:

$$\phi_{\mathcal{P}}^{\text{causal}} := \bigwedge_{\substack{e \in E \\ e' \in {}^\bullet({}^\bullet e \cup \underline{e})}} (\mathsf{e} \to \mathsf{e}') \qquad\qquad \phi_{\mathcal{P}}^{\text{sym}} := \bigwedge_{c \in C} \text{AMO}(c^\bullet),$$

where $\text{AMO}(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ is satisfied iff *at most one* of $\mathsf{x}_1, \ldots, \mathsf{x}_n$ is satisfied(see Section 5.1). $\phi_{\mathcal{P}}^{\text{asym}}$ ensures that $\mathcal{C}$ is free of $\nearrow$-cycles; the details come in Section 4.1. $\phi_{\mathcal{P}}^{\text{mark}}$ characterizes supersets of the marking $m$ reached by $\mathcal{C}$:

$$\phi_{\mathcal{P}}^{\text{mark}} := \bigwedge_{\substack{c \in B \\ p = f(c) \\ \{e\} = {}^\bullet c}} \left( \left( \mathsf{e} \wedge \bigwedge_{e' \in c^\bullet} \neg \mathsf{e}' \right) \to \mathsf{p} \right)$$

Finally, $\phi_{\mathcal{P}}^{\text{dis}}$ ensures that $m$ is indeed deadlocked in $N$:

$$\phi_{\mathcal{P}}^{\text{dis}} := \bigwedge_{t \in T} \bigvee_{p \in {}^\bullet t \cup \underline{t}} \neg \mathsf{p}$$

Notice that a variable $\mathsf{p}$ may be true even if $p \notin m$. However, such an assignment can only serve to *hide* a deadlock, so this encoding is safe.

### 4.1 Asymmetric conflict loops

We now explain $\phi_{\mathcal{P}}^{\text{asym}}$, which ensures that $\nearrow_{\mathcal{C}}$ is acyclic (for convenience, we equate a relation with a directed graph in the natural way). Symmetric conflicts form cycles of length 2 in $\nearrow$ and are efficiently handled by the AMO constraints of $\phi_{\mathcal{P}}^{\text{sym}}$. In a Petri net, these are the only cycles that can occur. However, in a c-net there may also be cycles in the relation $R := \lessdot \cup \nearrow\!\!\!\!\nearrow$. We show now that they occur naturally in well-known examples:

Consider Fig. 2, which shows the beginning of an unfolding of Dekker's mutual-exclusion algorithm [19] (only some events of interest are shown). In the beginning, both processes indicate their interest to enter the critical section by raising their flag (events $e_1, f_1$). They then check whether the flag of the other process is low (events $e_2, f_2$) and if so, proceed ($e_3$) and possibly repeat ($e_4, e_5$). If both processes want to enter the critical section ($f_2'$), some arbitration happens (not displayed). Two conflict cycles in this example are $e_1 \lessdot e_2 \nearrow\!\!\!\!\nearrow f_1 \lessdot f_2 \nearrow\!\!\!\!\nearrow e_1$ and $f_1 \lessdot f_2' \nearrow\!\!\!\!\nearrow e_3 \lessdot e_4 \lessdot e_5 \nearrow\!\!\!\!\nearrow f_1$.

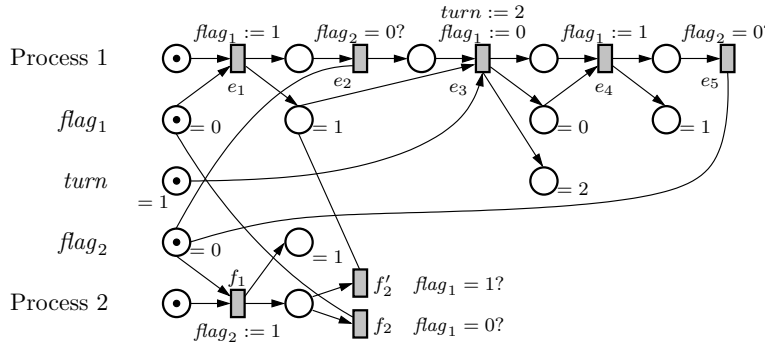


**Fig. 2.** Partial unfolding of Dekker's algorithm algorithm with asymmetric cycles.

Several encodings have been proposed in the literature for acyclicity constraints, including transitive closure and ranks (see, e.g., [4]). In the ranking method, one introduces for each event $e$ additional boolean variables that represent an integer up to $r$ (the so-called *rank* of $e$), where $r$ is a large enough number. Then, for each pair $(e, f) \in R$, one introduces a clause $(\mathsf{e} \wedge \mathsf{f}) \rightarrow [\![\mathrm{e} < \mathrm{f}]\!]$, where $[\![\mathrm{e} < \mathrm{f}]\!]$ is an additional variable that, if true, forces the rank of $e$ to be less than the rank of $f$. Naturally, this clause is only necessary if $e$ and $f$ are in the same strongly connected component (SCC) of $R$.

A lower bound for $r$ is the length of the longest chain in $\nearrow$ that does not contain a cycle; however, finding the latter is itself an NP-complete problem. A simple upper bound for $r$ is the size of the largest SCC of $R$. To further reduce this upper bound, one can exploit the fact that $\mathcal{C}$ is causally closed and that every cycle in $R$ contains at least two edges stemming from $\nearrow\!\!\!\!\nearrow$. Consider the relation $R' := \{\, (e, g) \mid \exists f, h : e \nearrow\!\!\!\!\nearrow f \leq g \nearrow\!\!\!\!\nearrow h \,\}$. One can easily see that any causally closed set of events contains a cycle in $R$ iff it contains a cycle in $R'$, so $r$ can be bounded by the largest SCC of $R'$ instead.

On the other hand, $R'$ may actually contain more pairs than $R$, and computing $R'$ may take quadratic time in $|E|$. So instead, we reduce the size of $R$ by a less drastic method that can run in linear time: An event $e$ is eliminated from $R$ by fusing its incoming and outgoing edges in $R$ only if (i) $e$ is not the source of a $\nearrow\!\!\!\!\nearrow$-edge and (ii) fusing the edges and eliminating $e$ will not increase the number of edges in $R$.

Fig. 2 demonstrates another important point. The figure contains two different cycles, both of which contain $f_1$. Thus, all events in Fig. 2 belong to the same SCC in $R$. Indeed, we observe in our experiments that the SCCs of $R$ tend to be large, often composed of thousands of events, but consist of many short, interlocking cycles. This suggests that an upper bound for $r$ better than the size of $R$, even after reduction, may still be feasible. We therefore suggest another trick: first, check for deadlock while omitting $\phi_{\mathcal{P}}^{\mathrm{asym}}$ from $\phi_{\mathcal{P}}^{\mathrm{dead}}$ altogether. This may result in false positives, i.e. a set of events leading to a deadlocked marking that is not actually reachable because it contains a cycle in $\nearrow$. If the SAT solver

comes up with such a spurious deadlock, repeat with $\phi_{\mathcal{P}}^{\text{asym}}$ properly included. The experiments concerning these points are discussed in Section 5.1.

### 4.2 Reduction of stubborn events

In this section, we discuss an optimization that palliates a problem of SAT checkers. Consider the occurrence net shown in Fig. 3. If event $e_1$ fires, then nothing can prevent $e_2$, $e_3$, $e_4$, and $e_5$ from firing. Thus, any configuration leading to a deadlock must either contain all five events or none of them. However, $e_1$ is not guaranteed to fire due to the white event that consumes from its context.

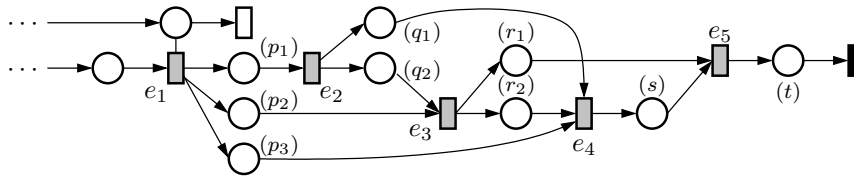


**Fig. 3.** Stubborn events.

In SAT solving, the value of a variable that is either known or has been tentatively decided is *propagated* to simplify other clauses [6]. Thus, in the SAT encoding for Fig. 3 (see [21] for more details), a SAT solver can immediately decide that no deadlock configuration may contain $e_5$ when the black event is a cutoff. This propagation is handled very efficiently by modern solvers, and there is no gain in emulating this behaviour while generating the SAT encoding.

However, unit propagation in our encoding is not able to detect that $\mathtt{e_3}$ and $\mathtt{e_4}$ are logical implications of $\mathtt{e_1}$. Even when a solver tentatively sets $\mathtt{e_1}$ to true, unit propagation only infers that $e_2$ must also be true, but not $e_3$ or $e_4$. It takes another decision, e.g. for $e_3$ or $e_4$, to derive a contradiction and, depending on the solver, possibly multiple steps to decide that $\mathtt{e_1}$ must necessarily be false.

On the other hand, such information is easy to detect on the unfolding structure, and we shall modify the proposed SAT encoding in these cases. Let us call *stubborn* any event $e$ satisfying $({}^\bullet e \cup \underline{e})^\bullet = \{e\}$. Intuitively, once all events preceding $e$ have fired, then firing $e$ is unavoidable to find a deadlock. In Fig. 3, events $e_2, e_3, e_4, e_5$ are all stubborn.

Indeed, consider any deadlocked configuration $\mathcal{C}$ of $\mathcal{P}$, and let $e$ be any stubborn event verifying ${}^\bullet({}^\bullet e \cup \underline{e}) \subseteq \mathcal{C}$. Then either $e$ is in $C$ or it is enabled at $cut(\mathcal{C})$, since $\mathcal{C}$ contains all events preceding $e$. But the latter is not possible because $\mathcal{C}$ is a deadlock, so $e$ must be in $C$, which proves that $e \in C$ iff ${}^\bullet({}^\bullet e \cup \underline{e}) \subseteq \mathcal{C}$ (the other direction follows from the fact that $\mathcal{C}$ is causally closed).

This suggests that we could substitute every occurrence of $\mathtt{e}$ by a conjunction of the variables associated to the predecessors of $e$. We denote by $E_s$ the set of stubborn events, and define inductively the set of *predecessors* of any event $e$ as $pred(e) := {}^\bullet({}^\bullet e \cup \underline{e}) \setminus E_s \cup \bigcup_{e' \in {}^\bullet({}^\bullet e \cup \underline{e}) \cap E_s} pred(e')$.

**Proposition 1.** *If $e$ is stubborn, then any deadlocked configuration $\mathcal{C}$ of $\mathcal{P}$ verifies that $e \in \mathcal{C}$ iff $pred(e) \subseteq \mathcal{C}$.*

**Corollary 1.** $\phi^{dead}_{\mathcal{P}} \equiv \phi^{dead}_{\mathcal{P}} \wedge \bigwedge_{e \in E_s} (\mathsf{e} \leftrightarrow \bigwedge_{e' \in pred(e)} \mathsf{e}')$

Corollary 1 can be exploited to modify $\phi^{\text{dead}}_{\mathcal{P}}$ in two ways: for every stubborn event $e$, (i) add a clause $\bigwedge_{e' \in \underline{e}^\bullet \cup {}^\bullet e)} \mathsf{e}' \rightarrow \mathsf{e}$, or (ii) substitute $\mathsf{e}$ by $\bigwedge_{e' \in pred(e)} \mathsf{e}'$. In our experiments, we chose method (ii), which eliminates the stubborn events from the encoding altogether. The resulting formula, after an initial unit propagation phase by the SAT solver, allows to immediately derive $\neg \mathsf{e}_1$. We note that in certain cases, this can increase the formula by a quadratic factor, see [21].

We briefly explain the changes to $\phi^{\text{dead}}_{\mathcal{P}}$ motivated by method (ii): $\phi^{\text{sym}}_{\mathcal{P}}$ is not affected because no stubborn event appears in any symmetric conflict, and neither is $\phi^{\text{dis}}_{\mathcal{P}}$. In $\phi^{\text{causal}}_{\mathcal{P}}$, however, clauses $\mathsf{e} \rightarrow \mathsf{e}'$ are replaced by $\mathsf{e} \rightarrow \mathsf{e}''$ for every $e'' \in pred(e)$. In a clause $\left(\mathsf{e} \wedge \bigwedge_{e' \in c^\bullet} \neg \mathsf{e}'\right) \rightarrow \mathsf{p}$ of $\phi^{\text{mark}}_{\mathcal{P}}$, we replace $\mathsf{e}$ by a conjunction over $pred(e)$. In principle, the same needs to be done for $\mathsf{e}'$. However, if $|c^\bullet| \geq 2$, then no event in $c^\bullet$ is stubborn, and nothing changes; but if $c^\bullet = \{e'\}$ is a singleton, and $e'$ is stubborn, then the clause is split into $|pred(e')|$ different clauses. For $\phi^{\text{asym}}_{\mathcal{P}}$, in a clause of the form $\mathsf{e} \wedge \mathsf{f} \rightarrow [\![\mathrm{e} < \mathrm{f}]\!]$, both $\mathsf{e}$ and $\mathsf{f}$ are replaced by conjunctions, if applicable; thus, the formula will still require ranks for $e$ and $f$ even if $\mathsf{e}$ or $\mathsf{f}$ are not present.

We remark that stubborn events are also treated specially in the stable-models encoding of [11]. While stable models are similar to SAT, the treatment in [11] is simpler; its analogue in propositional logic would not eliminate stubborn events from the formula nor allow to directly conclude that $e_1$ cannot be fired.

### 4.3 Additional simplification

We briefly mention some possible simplifications of the formula. First, for a place $p$, if $p^\bullet \cup \underline{p} = \emptyset$, then $\mathsf{p}$ does not appear in $\phi^{\text{dis}}_{\mathcal{P}}$ and can be omitted from $\phi^{\text{mark}}_{\mathcal{P}}$.

Secondly, for two conditions $c, d$, if $c^\bullet \subseteq d^\bullet$, then $\mathrm{AMO}(c^\bullet)$ is implied by $\mathrm{AMO}(d^\bullet)$ and can be omitted from $\phi^{\text{sym}}_{\mathcal{P}}$. Similarly, for two transition $t, u$ where ${}^\bullet t \subseteq {}^\bullet u$, disabledness of $t$ implies disabledness of $u$, so $u$ can be omitted from $\phi^{\text{dis}}_{\mathcal{P}}$. We return to this point in Section 5.1.

### 4.4 Reachability and Coverability

The SAT encoding can be easily modified to check reachability or coverability of a marking. For simplicity, the formulas given here are not directly in CNF.

For coverability, we want to check whether $N$ has a reachable marking $m$ such that $P_M \subseteq m$, where $P_M \subseteq P$. This requires the following modifications: $\phi^{\text{mark}}_{\mathcal{P}}$ still has the same intention but the sense of the implication is reversed; if a variable $\mathsf{p}$ is true we need to ensure that indeed some condition labelled by $p$ is marked in $\mathcal{C}$. We introduce additional variables $\mathsf{c}$ for some conditions $c$:

$$\phi^{\text{mark}}_{\mathcal{P}} := \bigwedge_{p \in P_M} \left(\mathsf{p} \rightarrow \bigvee_{f(c)=p} \mathsf{c}\right) \quad \wedge \quad \bigwedge_{f(c) \in P_M} \left(\mathsf{c} \rightarrow \left(\bigwedge_{e \in {}^\bullet c} \mathsf{e} \wedge \bigwedge_{e \in c^\bullet} \neg \mathsf{e}\right)\right)$$

Moreover, $\phi^{\text{dis}}_{\mathcal{P}}$ specifies reachability of $P_M$: $\phi^{\text{dis}}_{\mathcal{P}} := \bigwedge_{p \in P_M} \mathsf{p}$

For reachability, we want to check whether a given marking $m$ is reachable. Then, the variables representing the places must contain the exact marking reached by the event, which is achieved by replacing the one-sided implications of $\phi^{\text{mark}}_{\mathcal{P}}$ by equivalences. Moreover, $\phi^{\text{dis}}_{\mathcal{P}}$ needs to be changed to $\bigwedge_{p \in m} \mathsf{p} \wedge \bigwedge_{p \notin m} \neg \mathsf{p}$.

### 4.5 Bounded nets

We briefly sketch an extension to $k$-bounded nets. For deadlock checking, actually no modifications are needed because we require the preset and context of each transition to be a set. This is in the tradition of [1, 2, 22], where it helps to ease the presentation. However, if presets and contexts could be general multisets, then, for $p \in P$, one could replace the variable $\mathsf{p}$ by variables $\mathsf{p}^i$, where $1 \leq i \leq k$, with the meaning "$p$ carries at least $i$ tokens". Then one would modify $\phi^{\text{mark}}_{\mathcal{P}}$ to make $\mathsf{p}^i$ true if at least $i$ conditions with label $p$ are marked in $\mathcal{C}$, and $\phi^{\text{dis}}_{\mathcal{P}}$ requires that for each transition $t$ there exists some $p \in {}^\bullet t \cup \underline{t}$ such that $\mathsf{p}^i$ is false, where $i$ is the number of tokens in $p$ required by $t$. The extension for reachability is analogous, modulo the sense of the implication (cf. Section 4.4).

## 5 Experimental evaluation

In this section, we evaluate the SAT-based reduction proposed in Section 4. For this, we wrote a program that reads an unfolding prefix $\mathcal{P}$ generated by Cunf [20] and outputs the associated formula $\phi^{\text{dead}}_{\mathcal{P}}$ in DIMACS CNF format. As a SAT solver, we used the well-known tool MiniSat [6].

In Section 5.1, we first report on the effect of certain encoding variants and optimizations like those in Sections 4.1 to 4.3. In Section 5.2, we then compare against other unfolding-based methods, and we evaluate the effect of using c-nets rather than Petri nets. We concentrate on the aspect of deadlock checking; as pointed out in Section 4.4, the encoding for reachability is very similar.

### 5.1 Optimizations

Section 4 proposed several optimizations of the encoding. We now empirically evaluate their impact on the solving time. We employed as benchmarks the same set of safe nets that has previously been used in other papers of the literature on Petri net unfoldings, e.g. [11, 12, 22, 23]. For each Petri net $N$ in the set, we obtained a c-net $N'$ by substituting pairs of arcs $(p, t)$ and $(t, p)$ in $N$ by read arcs; we thus have a set of Petri nets and an alternative set of c-nets.

**Stubborn event elimination and subset reduction** Over the set of Petri nets shown in Table 2, we found that removal of stubborn events reduces the accumulated SAT solving time by 27%. When applied together with the subset optimization from Section 4.3, this grows to 30%. For c-nets, we measured a 14%

reduction when stubborn events are removed from the encoding *without* acyclicity constraints but only a 6% reduction if additionally the subset optimizations are applied. Experiments over the encoding *with* acyclicity were similar.

This suggest that removal of stubborn events has a positive impact on performance, while subset optimization has very limited, even negative impact. For the following, we applied only the stubborn event optimization.

**AMO constraint** The constraint $\mathrm{AMO}(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ in $\phi^{\mathrm{sym}}_{\mathcal{P}}$ can be trivially encoded by $\bigwedge_{1 \leq i < j \leq n} (\neg \mathsf{x}_i \vee \neg \mathsf{x}_j)$. However, this *pairwise encoding* is quadratic, and the SAT performance suffered for examples with large conflict sets.

A survey of better encodings can be found in [3]. Our tool uses a *k-tree* encoding, that introduces $\mathcal{O}(n)$ additional variables and adds $\mathcal{O}(n)$ clauses, see [21]. We observed an overall improvement when replacing the pairwise with the $k$-tree encoding. The accumulated SAT solving time on our benchmarks under values of $k = 2, \ldots, 8$ was minimal for $k = 4$. Experiments over c-nets on the encoding suggested $k = 4$ as a good candidate, as well. We therefore used 4-tree encodings in $\phi^{\mathrm{sym}}_{\mathcal{P}}$ for the following experiments.

**Acyclicity checking** Section 4.1 explained that $\phi^{\mathrm{asym}}_{\mathcal{P}}$ encodes cycle-freeness of configuration $\mathcal{C}$ w.r.t. the relation $R = < \cup \nearrow$. We investigated three encodings suggested in [4]: transitive closure, unary ranks, and binary ranks. The latter clearly outperformed the others. In the binary rank encoding, every event is associated with a rank, i.e. an integer up to some bound $r$, that is represented by $\lceil \log_2 r \rceil$ boolean variables. Constraints of the form $[\![\mathsf{e} < \mathsf{f}]\!]$ ensure that the rank of event $e$ is less than the rank of event $f$ if $(e, f) \in R$. If $n$ is the number of events in $\mathcal{P}$, the resulting SAT encoding is of size $\mathcal{O}(n^2 \log n)$.

Moreover, Section 4.1 proposed a method to reduce the size of $R$. Table 1 shows the size of the direct asymmetric conflict relation before and after this reduction for some c-nets unfoldings with at least one cycle in $R$. More precisely, we show the size of the largest SCC (in most examples there is in fact only one non-trivial SCC). In average, the method proposed eliminates 66% of the nodes and 26% of the edges, seeming thus to be more effective at reducing the number of nodes rather than the number of edges, wich in turn becomes a reduction in the number of variables rather than the number of clauses of the encoding.

However, in some examples, the remaining SCCs are still rather large, on the order of tens of thousands of events, and in these cases $\phi^{\mathrm{asym}}_{\mathcal{P}}$ negatively impacts the running time. We therefore implemented a two-stage approach, in which the first stage simply omits $\phi^{\mathrm{asym}}_{\mathcal{P}}$ from the formula. Only when this first stage yields a false positive, a second stage with $\phi^{\mathrm{asym}}_{\mathcal{P}}$ is used to obtain a definitive result. This approach was very successful: in over 100 different nets from various sources that we tried, only 2 (small) nets yielded a false positive. The experiments presented in the following use this two-stage approach.

**SAT-solver settings** MiniSat allows to change aspects of the SAT-solving algorithm, such as decision variables, default polarity etc. We attempted to tweak

| Net | Before reduction Nodes | Edges | After reduction Nodes | Edges | Ratio after/before Nodes | Edges |
|---|---|---|---|---|---|---|
| bds_1.sync | 192 | 271 | 27 | 52 | 0.14 | 0.19 |
| byzagr4_1b | 3197 | 64501 | 2348 | 61088 | 0.73 | 0.95 |
| q_1.sync | 189 | 4095 | 126 | 4032 | 0.67 | 0.98 |
| bds_1.fsa | 66 | 89 | 9 | 16 | 0.14 | 0.18 |
| dme11 | 8745 | 44968 | 4918 | 40301 | 0.56 | 0.90 |
| rw_2w1r | 1766 | 8877 | 915 | 7447 | 0.52 | 0.84 |

**Table 1.** Reduction of the asymmetric-conflict relation.

these settings in order to exploit knowledge about the problem domain, but without obtaining significant improvements. More details are given in [21].

### 5.2 Comparisons

In [15], Khomenko and Koutny compared three versions of their deadlock checking method, implemented in the tool CLP, against the methods by McMillan [17], Melzer and Römer [18], and Heljanko [11]. In their benchmarks, the first version of their algorithm[2] outperformed the other methods on almost all examples. We experimentally confirmed this conclusion. Moreover, we learnt of an unpublished SAT-based tool by Khomenko which is said to be slower than CLP.[3] We therefore compare our technique with the first method of CLP.[4]

We discuss two families of examples: a standard suite of benchmarks known from the unfolding literature (see Section 5.1), and another family encoding networks of logic gates. The first family does not specifically exploit the features of c-nets; here the savings are not dramatic but still significant. In the second family, c-nets lead to large time savings.

Table 2 presents the results on the aforementioned standard suite. We used MOLE [24] to produce finite complete prefixes of the Petri nets and CUNF [20] to do the same for c-nets.[5] The running times for MOLE and CUNF are given in the respective columns, the number of events and conditions of the two prefixes is indicated in the columns $|E|$ and $|B|$. For Petri nets, we also give the running times of CLP, and the running time of MINISAT in our encoding on the Petri net. For c-nets we provide the running times of MINISAT with the settings discussed in Section 5.1. Times are given in seconds and represent averages over 10 runs.

[2] Column *std* in Tables 1 and 2 in [15].

[3] According to the author, V. Khomenko.

[4] All experiments have been performed using CUNF v1.4, MOLE v1.0.6, both compiled with `gcc` 4.4.5, version 301 of CLP, and MINISAT v2.2.0. Our machine has twelve 64bit Intel Xeon CPUs, running at 2.67GHz, 50GB RAM and executes Linux 2.6.32-5.

[5] The running times of MOLE and CUNF are comparable on Petri nets, but MOLE produces prefixes in a format suitable for CLP.

| | | Petri net unfolding | | | | | c-net unfolding | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Mole | | | clp | SAT | Cunf | | | SAT |
| Net | Res. | Time | $\|E\|$ | $\|B\|$ | Time | Time | Time | $\|E\|$ | $\|B\|$ | Time |
| bds_1.sync | L | 0.58 | 12900 | 37306 | 0.04 | 0.01 | 0.14 | 1830 | 2771 | <0.01 |
| byzagr4_1b | L | 3.71 | 14724 | 42276 | 0.53 | 0.26 | 3.25 | 8044 | 17603 | 0.19 |
| dme11 | L | 6.56 | 9185 | 31186 | 0.60 | 0.28 | 10.86 | 9185 | 16710 | 0.25 |
| dpd_7.sync | L | 1.21 | 10354 | 29939 | 0.10 | 0.18 | 1.09 | 10354 | 21359 | 0.02 |
| ftp_1.sync | L | 45.37 | 91730 | 275099 | 1.13 | 0.38 | 26.85 | 50928 | 96617 | 0.05 |
| furnace_4 | L | 37.44 | 114477 | 264823 | 1.29 | 0.19 | 19.11 | 94413 | 147438 | 0.12 |
| rw_12.sync | L | 3.95 | 98361 | 295152 | 0.08 | 0.02 | 3.96 | 98361 | 196796 | 0.02 |
| rw_1w3r | L | 0.30 | 15432 | 28207 | 0.11 | 0.22 | 0.36 | 14521 | 24174 | 0.40 |
| rw_2w1r | L | 0.22 | 9363 | 18575 | 0.04 | 0.34 | 0.32 | 9363 | 15304 | 0.58 |
| elevator_4 | D | 2.58 | 16856 | 47743 | 0.24 | 0.03 | 1.51 | 16856 | 28593 | 0.06 |
| key_4 | D | 1.68 | 69600 | 139206 | 0.07 | 0.08 | 2.07 | 4754 | 7862 | <0.01 |
| mmgt_4.fsa | D | 1.16 | 46902 | 92940 | 0.02 | 0.04 | 1.17 | 46902 | 92076 | 0.05 |
| q_1.sync | D | 1.76 | 10716 | 30087 | <0.01 | 0.02 | 1.54 | 10716 | 20567 | 0.01 |
| $\sum$ | | 106.52 | | | 4.25 | 2.05 | 72.23 | | | 1.75 |

**Table 2.** Comparison of deadlock-checking methods; the Res(ult) is L(ive) or D(ead)

We do not provide the translation times to generate linear equation systems (for clp) or SAT formulas (for MiniSat). Those times would not be very representative since both translators are suboptimal; our own translator to SAT is in a preliminary stage. Also, there is no reason to suspect that the translation times for the linear equations of [15] and SAT, when optimized, would be very different, and we expect such optimized times to be fractions of a second.

Compared to clp, SAT checking performs well over Petri nets, solving the problems twice as fast on aggregate. Concerning the comparison of SAT checking between Petri nets and c-nets, we obtain another advantage of 13% for deadlock verification. More significantly, the time for generating c-net unfoldings is 30% less than for Petri nets. This advantage is not huge, but recall that these benchmarks are already favourable examples for Petri net unfoldings and were not specifically designed to exploit the advantages of c-nets. The two-stage approach was essential for performance: while the acyclic constraints had a big impact only on a few examples (notably `byzagr4_1b,dme,rw*`), that effect would have more than nullified the advantage of faster unfolding times.

We now present a class of nets in which read arcs have natural advantages: the encoding of asynchronous circuits of logic gates as Petri nets, one of the motivations originally mentioned by McMillan [17]. In this encoding, the signals, i.e. the inputs and outputs of each gate, are modelled with two places for indicating whether the signal is high (1) or low (0). The outputs change as a function of

*reading* the inputs. Fig. 4 (a) shows an AND-gate and its encoding as a c-net fragment.

To illustrate the benefits that c-nets enjoy here, we discuss a simple experiment. We consider a grid of $n := k \times k$ AND-gates, shown in Fig. 4 (b) for $k = 3$. The inputs for the AND-gates are at the left and top of the figure, and outputs propagate to the right and towards the bottom. Inputs may switch freely between high and low. We encoded such grids into c-nets; additionally, we replaced read arcs with arrow loops to obtain equivalent Petri nets (so called *plain encodings*). We then used CUNF to construct complete unfolding prefixes of the c-nets and their plain encodings, and observed that signal changes may be propagated to the bottom right in many different orders, which are distinguished by Petri-net unfoldings but not by c-net unfoldings. Hence, unfoldings of the plain nets were of exponential size in $n$, while the contextual ones were linear. Moreover, CUNF built the latter ones in time $\mathcal{O}(n^3)$, see Fig. 4 (c). The verification method for c-nets herein presented allows to profit from the reduced unfolding time.

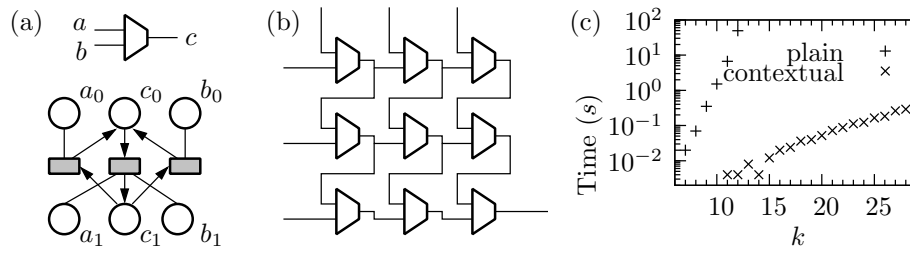


**Fig. 4.** (a) Encoding of a logical AND-gate; (b) grid of AND-gates; (c) unfolding times

## 6 Conclusions

We presented verification algorithms based on c-net unfoldings. The twofold advantages over previous work are the overall performance of the SAT encoding, and that c-nets allow to profit from faster unfolding procedures and/or faster verification on the resulting unfolding prefixes. The latter result was not a foregone conclusion due to the richer structure of c-net unfoldings, in particular the presence of cycles and histories.

We studied optimizations of the encoding, concentrating on optimizations on the net level, while leaving optimizations on the logical level to the SAT solver.

An interesting future direction of work would be to extend the verification algorithms to a richer set of properties. E.g., LTL model-checking for Petri nets has been investigated in [7], but the trace logics investigated by Diekert and Gastin [5] and others seem like another natural choice.

*Acknowledgements:* The authors would like to thank Keijo Heljanko, Victor Khomenko, Paolo Baldan, and the referees for helpful hints and discussions.

## References


1. Baldan, P., Bruni, A., Corradini, A., König, B., Schwoon, S.: On the computation of McMillan's prefix for contextual nets and graph grammars. In: Proc. ICGT'10. LNCS, vol. 6372, pp. 91–106 (2010)
2. Baldan, P., Corradini, A., König, B., Schwoon, S.: McMillan's complete prefix for contextual nets. ToPNoC 1, 199–220 (2008), LNCS 5100
3. Chen, J.: A new SAT encoding of the at-most-one constraint. In: Proc. Constraint Modelling and Reformulation (2010)
4. Codish, M., Genaim, S., Stuckey, P.J.: A declarative encoding of telecommunications feature subscription in SAT. In: Proc. PPDP. pp. 255–266. ACM (2009)
5. Diekert, V., Gastin, P.: From local to global temporal logics over Mazurkiewicz traces. Theoretical Computer Science 356(1-2), 126–135 (May 2006)
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. SAT. pp. 502–518. LNCS 2919 (2003)
7. Esparza, J., Heljanko, K.: Implementing LTL model checking with net unfoldings. In: Proc. SPIN. LNCS, vol. 2057, pp. 37–56 (2001)
8. Esparza, J., Heljanko, K.: Unfoldings - A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science, Springer (2008)
9. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Formal Methods in System Design 20, 285–310 (2002)
10. Esparza, J., Schröter, C.: Unfolding based algorithms for the reachability problem. Fund. Inf. 47(3-4), 231–245 (2001)
11. Heljanko, K.: Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. Fund. Inf. 37(3), 247–268 (1999)
12. Khomenko, V.: Model Checking Based on Prefixes of Petri Net Unfoldings. Ph.D. thesis, School of Computing Science, Newcastle University (2003)
13. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged processes – a new condensed representation of Petri net behaviour. Act. Inf. 43(5), 307–330 (2006)
14. Khomenko, V., Koutny, M.: LP deadlock checking using partial order dependencies. In: Proc. CONCUR. pp. 410–425. LNCS 1877 (2000)
15. Khomenko, V., Koutny, M.: Verification of bounded Petri nets using integer programming. Formal Methods in System Design 30(2), 143–176 (2007)
16. Khomenko, V.: Punf, `homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/`
17. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Proc. CAV. pp. 164–177. LNCS 663 (1992)
18. Melzer, S., Römer, S.: Deadlock checking using net unfoldings. In: Proc CAV. LNCS, vol. 1254, pp. 352–363 (1997)
19. Raynal, M.: Algorithms for Mutual Exclusion. MIT Press (1986)
20. Rodríguez, C.: Cunf, `http://www.lsv.ens-cachan.fr/~rodriguez/tools/cunf/`
21. Rodríguez, C., Schwoon, S.: Verification of Petri Nets with Read Arcs. Tech. Rep. LSV-12-12, LSV, ENS de Cachan (2012)
22. Rodríguez, C., Schwoon, S., Baldan, P.: Efficient contextual unfolding. In: Proc. Concur. LNCS, vol. 6901, pp. 342–357 (September 2011)
23. Schröter, C.: Halbordnungs- und Reduktionstechniken für die automatische Verifikation von verteilten Systemen. Ph.D. thesis, Universität Stuttgart (2006)
24. Schwoon, S.: Mole, `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`
25. Schwoon, S., Rodríguez, C.: Construction and SAT-based verification of contextual unfoldings. In: Proc. DCFS. pp. 34–42. LNCS 6808 (2011), extended abstract