Model Checking Using Net Unfoldings*

Javier Esparza
Institut für Informatik, Universität Hildesheim
Samelsonplatz 1, W-3200 Hildesheim, Germany.

Abstract. In [3], McMillan described a technique for deadlock detection based on net unfoldings. We extend its applicability to the properties of a temporal logic with a possibility operator. The algorithm is based on Linear Programming. It compares favourably with other algorithms for the class of deterministic concurrent systems.

1 Introduction

Model checking has become a well established paradigm for verifying that a concurrent program satisfies a temporal logic formula. It views the program as a structure on which to interpret the considered logic and evaluates the formula on this structure.
In most work on model checking, the structures are some sort of transition system obtained representing concurrency by arbitrary interleaving. It has been observed that this contributes to a state explosion problem, and that avoiding the enumeration of all interleavings could lead to more efficient algorithms.
Several researchers have proposed such algorithms. Some of them use partial order notions to reduce the size of the state space, such as the stubborn sets method of Valmari (see, for instance [12]) or the trace automaton method of Godefroid and Wolper (see, for instance, [5]). Others work directly on partial order structures, such as the behaviour machines of Probst and Li [10] and the Petri net unfoldings of McMillan [8]. We are particularly interested in the latter. McMillan's method is based on the net unfoldings introduced by Nielsen, Winskel and Plotkin in [9] as a partial order semantics of Petri nets (closely related to event structures) and further studied by Engelfriet in [3]. For verification purposes, these unfoldings have the problem of being infinite even for systems with a finite number of states. McMillan shows how to construct a finite prefix of the unfolding large enough to be able to detect deadlocks.
In this paper, we make deeper use of the theory of unfoldings to extend M-cMillan's approach to model checking: we propose a verification algorithm for a logic closely related to S_4 [7], which extends propositional logic with a possibility operator, and permits to express safety properties such as the reachability of a

*This work was partially supported by the Esprit Basic Research Action CALIBAN

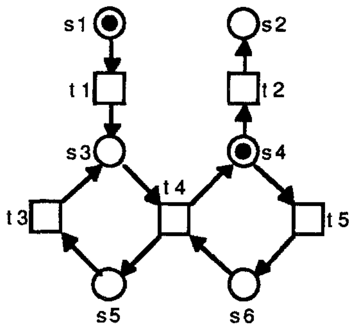

Figure 1: A finite 1-safe system.

state or the liveness of a transition.
Since the unfolding does not contain any explicit representation of the states of the system, our algorithm is very different from the traditional traversing algorithms of the state space. It uses Linear Programming to identify certain particularly important states.
Our algorithm is adequate for deterministic or nearly deterministic concurrent net systems, with applications to the design of asynchronous circuits. In the deterministic case (no conflicts) our algorithm is polynomial in the size of the prefix. For the class of conflict-free net systems [6,13] it is even polynomial in the size of the system, whereas the algorithms of [5,10,12] are either exponential or not applicable.
This paper is a shortened version of [4], where the proofs omitted here can be found. The paper generalises results of [1]. There, a model checker was given for the class of 1-safe Petri nets in which places have at most one input transition and at most one output transition. Here we generalise this model checker to arbitrary 1-safe Petri nets.

The paper is organised as follows. Section 2 is devoted to basic notations and results. Section 3 introduces the syntax and semantics of our logic. Section 4 shows how to reduce the model checking problem to two simpler problems. Algorithms for these problems are given in sections 5 and 6. Finally, Section 7 considers the case of deterministic concurrent systems.

2 Basic Notions

We assume that the reader is familiar with the basic notions and notations of Petri nets, as given for instance in [2]. We use finite 1-safe Petri nets as system models. We denote a Petri net by $\Sigma = (S, T, F, M_0)$, where (S, T, F) is a net and M_0 its initial marking. A place of a 1-safe Petri net can contain at most one token. In the sequel, we call 1-safe Petri nets 1-safe systems, or just systems. Figure 1 shows a 1-safe system.

Our execution model are Engelfriet's branching processes [3]. Branching processes are unfoldings of net systems containing information about both concurrency and conflicts.
Let (S,T,F) be a net and x_1, $x_2 \in S \cup T$. x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist distinct transitions t_1, $t_2 \in T$ such that $|{}^\bullet t_1| \cap |{}^\bullet t_2| \neq \emptyset$, and both (t_1, x_1), (t_2, x_2) belong to the reflexive and transitive closure of F. We define now occurrence nets, which are the nets underlying branching processes. To differentiate the system and execution levels, places of occurrence nets are called *conditions*, and their transitions are called *events*.
A net (B,E,F) is called *occurrence net* if

(i) for every $b \in B$, $|{}^\bullet b| \leq 1$,

(ii) the transitive closure of F is irreflexive, and

(iii) no event $e \in E$ is in conflict with itself (i.e., not $e \# e$).

$Min(N)$ denotes the set of minimal elements of $B \cup E$ with respect to the transitive closure of F.
A *branching process* of a system $\Sigma = (S,T,F,M_0)$ is a pair $\beta = (N',p)$ where $N' = (B,E,F)$ is an occurrence net and $p: B \cup E \rightarrow S \cup T$ a labelling function of N' satisfying certain properties that make (N',p) an unfolding of the system [1].
Figure 2 shows a branching process of the system of Figure 1. The names of the events have been written within the boxes, while the transitions associated to them have been written close to them. The names of the conditions have been omitted to keep the picture simple.
In [3] an *approximation relation* is defined between branching processes. The exact definition is not necessary for the purpose of this paper. Intuitively, β_1 approximates β_2 if β_1 is isomorphic to an initial part of β_2. It was proved in [3] that a system has a unique maximal branching process (up to isomorphism) with respect to the approximation relation. The maximal branching process of the system of Figure 1 is infinite. Loosely speaking, it consists of a periodic repetition of the initial part of the branching process of Figure 2 obtained 'cutting' the net by a vertical line just to the right of the event e_4.

Let (B,E,F) be an occurrence net. By definition, the transitive closure of F is a partial order. We denote it by $\prec$. $\preceq$ denotes the reflexive and transitive closure of F. Given $x \in B \cup E$ and $X \subseteq B \cup E$, we say $x \prec X$ if there exists $x' \in X$ such that $x \prec x'$.
A subset $E' \subseteq E$ is a *configuration* if it is left-closed with respect to $\preceq$ and no two elements of E' are in conflict. In the branching process of Figure 2, the set $\{e_1, e_2\}$ is a configuration, while the sets $\{e_4\}$ and $\{e_1, e_3\}$ are not ($\{e_4\}$ is not left-closed, and $e_1 \# e_3$). $X \subseteq B$ is a co-set if $\forall x_1, x_2 \in X: \neg(x_1 \prec x_2) \wedge \neg(x_2 \prec x_1)$. A maximal co-set with respect to set inclusion is called a cut. It is well known [2,3] that if c is a cut of a branching process $\beta = (N',p)$, then $p(c)$ is a reachable marking of Σ. The converse holds for the maximal branching process.

[1]The exact definition is not relevant for this paper. It can be found in [3].

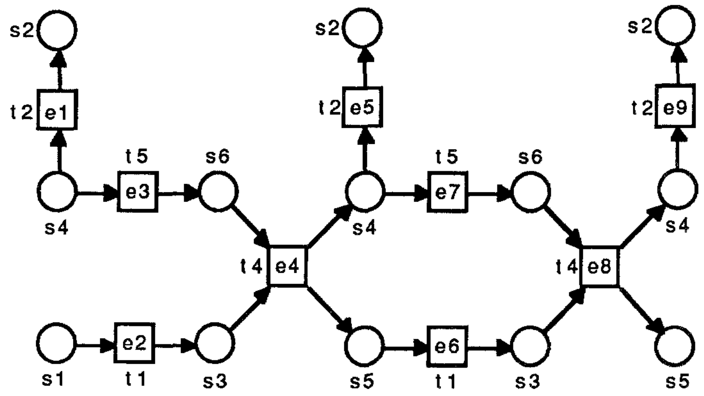

Figure 2: A branching process of the system of Figure 1

We associate to a configuration C a set of conditions $Cut(C)$ in the following way:

$$Cut(C) = (Min(N) \cup C^{\bullet}) \setminus {}^{\bullet}C$$

It is easy to prove that Cut is a bijection between the set of finite configurations of a branching process and its set of cuts.

Cuts will be used in the definition of the satisfaction relation of our logic. However, we shall mainly work with finite configurations, because they have a simpler mathematical structure, and cuts can always be retrieved via the Cut mapping.

3 A Modal Logic for 1-Safe Systems

We define in this section the syntax and semantics of a modal logic tailored for finite 1-safe systems.
We fix for the rest of the paper a finite 1-safe system $\Sigma = (S, T, F, M_0)$ such that every element of $S \cup T$ has a nonempty preset or a nonempty postset.
As pointed out in Section 2, Σ has a unique maximal branching process up to isomorphism. We fix a representative of the isomorphism class, denoted by

$$\beta_m = (B_m, E_m, F_m, p_m).$$

The logic is essentially S_4 [7] interpreted over the set of configurations of β_m, with elementary propositions tailored for Petri nets. The logic extends propositional logic with a possibility operator. The set of formulas over Σ is generated by the following grammar:

$$\phi ::= \mathbf{true} \mid s \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \Diamond\phi$$

The operator $\Box$ is defined by $\Box = \neg\Diamond\neg$. The logic S_4 is interpreted over a set of worlds having a preorder structure (or stronger). In our case, it is the set of

finite configurations of β_m.
The satisfaction relation $\models$ is defined as usual for the propositional connectives. Moreover, for a finite configuration C we have:

$C \models s$ if $s \in p_m(Cut(C))$.
$C \models \Diamond\phi$ if there exists a finite configuration $C' \supseteq C$ such that $C' \models \phi$.

Finally, we say that Σ satisfies ϕ, also denoted by $\Sigma \models \phi$, if $\emptyset \models \phi$.
Loosely speaking, $C \models s$ if after the occurrence of the events of C a marking is reached in which the place s contains a token. For instance, in the branching process of Figure 2, $\{e_1, e_2\} \models s_2$. We have $C \models \Diamond\phi$ if C can be extended to a configuration C' (lying therefore in the 'future' of C) such that C' satisfies ϕ. Notice that $\Diamond\phi$ means 'possibly ϕ' and not 'eventually ϕ'.
The logic permits one to express safety properties such as the reachability of a marking (the system of figure 1 satisfies $\Diamond(\neg s_1 \wedge s_2 \wedge \neg s_3 \wedge s_4 \wedge \neg s_5 \wedge \neg s_6)$ iff the marking $\{s_2, s_4\}$ is reachable), the liveness of a transition (the system satisfies $\Box\Diamond s_4$ iff transition t_5 is live) or the fact that a marking is a home state (the initial marking can always be reached again iff the system satisfies $\Box\Diamond(s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge s_4 \wedge \neg s_5 \wedge \neg s_6)$).
We study in the sequel the model checking problem for this logic, i.e. whether $\Sigma \models \phi$ for a formula ϕ. It is immediate to see that the model checking problem reduces to the problem for formulas of the form $\Diamond\phi$ (formulas without modalities can be easily checked using directly the definition of $\models$).
We denote by $Sat(\phi)$ the set of configurations of β_m that satisfy ϕ. It follows easily from the definitions that $\Sigma \models \Diamond\phi$ iff $Sat(\phi) \neq \emptyset$. Therefore, the model checking problem further reduces to deciding for a formula ϕ if $Sat(\phi)$ is empty.

4 The Finite Prefix

The maximal branching process β_m may be infinite, and therefore unsuitable for verification. We define in this section a finite prefix of it. We say that a branching process $\beta = (B, E, F, p)$ of Σ is a prefix of β_m if β approximates β_m, and moreover $B \subseteq B_m$ and $E \subseteq E_m$.
The finite prefix β_f is chosen to ensure that all reachable markings of Σ are represented by cuts of β_f. This is necessary if we wish to base a model checker on β_f; otherwise, for the formula ϕ corresponding to the reachability of a marking not represented in β_f, we would have $Sat(\phi) \neq \emptyset$ but we could hardly decide it using information from β_f only.
The branching process β_f was defined and used for deadlock detection by M-cMillan in [8]. Its definition is based on the notion of set of causes of an event. Given a branching process $\beta = (B, E, F, p)$ and $e \in E$, the set $[e] = \{e' \in E \mid e' \preceq e\}$ is the set of causes of e (in β). In the branching process of Figure 2, we have $[e_5] = \{e_2, e_3, e_4, e_5\}$.
It follows immediately from the definitions that the set of causes of an event is a finite configuration. Moreover, for every configuration C, either C does not contain e or it includes $[e]$.

Definition 4.1 *The prefix β_f [8]*

An event e of β_m is called a cut-off event if there exists a set of causes $[e'] \subset [e]$ (proper inclusion) such that

$$p_m(\,Cut([e'])\,) = p_m(\,Cut([e])\,)$$

(i.e. $[e']$ and $[e]$ correspond to the same reachable marking). β_f is the maximal prefix of β_m with respect to the approximation relation that contains no cut-off events.

E_f denotes the set of events of β_f. Given a cut-off event e, we denote by e^0 an arbitrarily selected event of $[e]$ such that

$$p_m(\,Cut([e^0])\,) = p_m(\,Cut([e])\,)$$

This event exists by the definition of cut-off event. ■ 4.1

The finiteness of β_f follows from the finiteness of the number of reachable markings of Σ [8]. Also, in [8] an algorithm is described for the construction of β_f. The algorithm searches for minimal cut-off events with respect to the relation $\prec$. When a minimal cut-off event is found, its succesors need not be explored, because they cannot be part of the prefix. The search is continued until all minimal cut-off events with respect to the relation $\preceq$ have been identified. The reader is referred to [8] for a detailed pseucode description of the algorithm and an evaluation of its performance.
The finite prefix of the system of Figure 1 is the one with $\{e_1, e_2, e_3, e_4, e_5, e_7\}$ as set of events as events (see Figure 3). The unique minimal cut-off event is e_6 (shaded in Figure 3). We take $e_6^0 = e_2$, because $e_2 \in [e_6]$, and

$$p(\,Cut([e_2])\,) = \{s_3, s_4\} = p(\,Cut([e_6])\,)$$

(these two cuts are represented in Figure 3 by straight lines that cross the conditions of the cut).

Define $\Uparrow e$ as the branching process containing all nodes x of β_m such that for some $b \in Cut([e])$, $b \preceq x$ (it is routine to check that $\Uparrow e$ is a branching process). Loosely speaking, $\Uparrow e$ contains the events and conditions after $[e]$, or, in other words, the "future" of the system from the marking corresponding to $Cut([e])$. Since $Cut([e])$ and $Cut([e^0])$ correspond to the same markings, $\Uparrow e$ and $\Uparrow e^0$ are isomorphic branching processes. Using this fact, we can prove that every reachable marking is represented in β_f. For every reachable marking M of Σ, there exists some configuration C such that $p_m(Cut(C)) = M$. If C is a configuration of β_f, then we are done. Otherwise, C contains some cut-off event e. Since $\Uparrow e$ is isomorphic to $\Uparrow e^0$, there exists another configuration C' after e^0 with the same associated marking, and containing *less* events than C, because $|[e]| > |[e^0]|$. If C' is not a configuration of β_f, then we iterate the procedure.

4.1 Shifts

We introduce some notions that allow us to formalise arguments like the one of the previous paragraph.

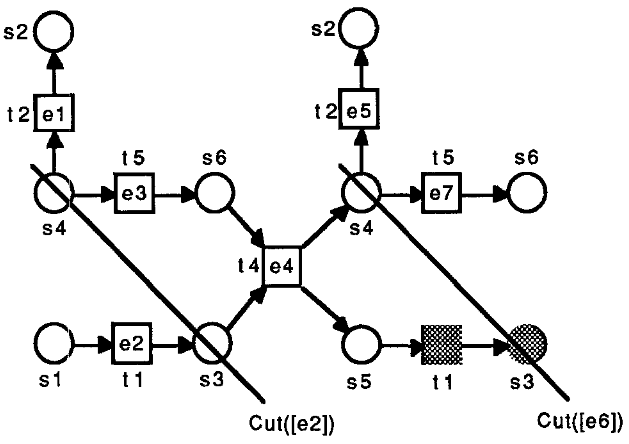

Figure 3: The finite prefix of the system of Figure 1

Definition 4.2 *Shift of a configuration*

Let e be a cut-off event of β_m and let I_e be an isomorphism from $\Uparrow e^0$ to $\Uparrow e$. Let C be a configuration of β_m.
The e-shift of C, denoted by $\mathcal{S}_e(C)$, is the following configuration:

$$\mathcal{S}_e(C) = \begin{cases} C & \text{if } e^0 \notin C \\ [e] \cup I_e(C \setminus [e^0]) & \text{if } e^0 \in C \end{cases}$$

■ 4.2

It is easy to prove that $\mathcal{S}_e(C)$ is a configuration, and therefore well defined. A configuration and its e-shift have associated the same reachable marking. In Figure 2, we have $\mathcal{S}_{e_6}(\{e_1, e_2\}) = [e_6] \cup I_{e_6}(\{e_1\}) = \{e_2, e_3, e_4, e_6\} \cup \{e_5\}$. Both $\{e_1, e_2\}$ and $\mathcal{S}_{e_6}(\{e_1, e_2\})$ have associated the marking $\{s_2, s_3\}$. We also have $|\mathcal{S}_e(C)| \geq |C|$, and the equality holds only if $\mathcal{S}_e(C) = C$.
For a set of configurations $\mathcal{C}$, we define $\mathcal{S}_e(\mathcal{C}) = \{\mathcal{S}_e(C) \mid C \in \mathcal{C}\}$.

We show that $Sat(\phi)$ can be generated from a subset of it, namely the set of configurations that satisfy ϕ and are contained in the finite prefix. Formally, we define $Sat_f(\phi) = Sat(\phi) \cap \mathcal{F}$, where $\mathcal{F}$ denotes the set of configurations of β_f This will reduce the model checking problem to deciding the emptyness of this subset.

Definition 4.3 *The mapping $\mathcal{S}$*

Let $\mathcal{C}$ be a set of configurations of β_m. The set of configurations $\mathcal{S}(\mathcal{C})$ is given by:

$$\mathcal{S}(\mathcal{C}) = \mathcal{C} \cup \bigcup_{e \in Off} \mathcal{S}_e(\mathcal{C})$$

where *Off* denotes the set of minimal cut-off events of β_m.
We define:

$$\mu\mathcal{S}.\mathcal{C} = \bigcup_{n \geq 0} \mathcal{S}^n(\mathcal{C})$$

■ 4.3

Clearly, $\mathcal{S}$ is a monotonic function on the complete partial order of sets of configurations of β_m. It is easy to see that $\mu\mathcal{S}.\mathcal{C}$ is the least fixpoint of $\mathcal{S}$ containing $\mathcal{C}$; this is the reason of the notation.
We can now state the first important result of the paper. Crudely speaking, it states that every configuration of $Sat(\phi)$ can be obtained by repeated shifting of some configuration of $Sat_f(\phi)$.

Theorem 4.4

Let ϕ be a formula. Then: $Sat(\phi) = \mu\mathcal{S}.Sat_f(\phi)$. ■ 4.4

In particular, we have $Sat(\phi) = \emptyset$ if and only if $Sat_f(\phi) = \emptyset$. That is, for every formula ϕ, if some configuration satisfies it, then some configuration of the finite prefix satisfies it as well. Therefore, the model checking problem can be reduced to deciding whether or not $Sat_f(\phi)$ is empty.

In Section 6, we shall show how to compute – using Linear Programming – the maximal elements of $Sat_f(\phi)$ for a simple subclass of formulas. In order to extend this to all the formulas of the logic, we obtain in the next section compositional equations for the maximal elements of $Sat_f(\phi)$.

4.2 Compositional equations

We introduce first a normal form for the formulas of our logic. It is a generalisation of the disjunctive normal form of propositional logic.

Definition 4.5 *Normal form*

A formula is in normal form if it is generated by the following grammar:

$$\begin{array}{rcl} \gamma & ::= & \textbf{true} \mid \textbf{false} \mid s \mid \neg s \mid \gamma \wedge \gamma \\ \phi & ::= & \gamma \mid \phi \wedge \Diamond\phi \mid \phi \wedge \neg\Diamond\phi \end{array}$$

■ 4.5

In the sequel, as was done in this definition, the symbol γ is used to denote conjunctions of literals.

Let $\phi_1 \equiv \phi_2$ be two formulas. ϕ_1 is equivalent to ϕ_2, denoted by $\phi_1 \equiv \phi_2$, if they have exactly the same models. We can prove the following proposition:

Proposition 4.6

Let ϕ be a formula. There exist formulas $\phi_1, \ldots, \phi_n$ in normal form such that $\phi \equiv \bigvee_{i=1}^{n} \phi_i$. ■ 4.6

By this result, $Sat(\phi) = \bigcup_{i=1}^{n} Sat(\phi_i)$ for a set of formulas $\{\phi_1, \ldots, \phi_n\}$ in normal form. It follows that deciding the emptyness of $Sat(\phi)$ for an arbitrary formula ϕ reduces to the problem of deciding the emptyness of $Sat(\phi)$ for a formula ϕ in normal form.
It must be remarked that the length of the conjunction of formulas in normal form equivalent to a given formula ϕ may be exponential in the length of ϕ. This makes our algorithm exponential in the length of the formula (however, as shown in [1], this cannot be avoided unless $P = NP$).

We show how to compositionally express $Sat(\phi)$ when ϕ is in normal form. First, we generalise the definition of $Sat(\phi)$ by introducing some more parameters.

Definition 4.7

Let C be a configuration and $\mathcal{C}$ a set of configurations. We say $C \leq \mathcal{C}$ if there exists $C' \in \mathcal{C}$ such that $C \subseteq C'$.
Let ϕ a formula and $\mathcal{C}_1$, $\mathcal{C}_2$ two sets of configurations. $C \in Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$ if $C \models \phi$, $C \not\leq \mathcal{C}_1$ and $C \leq \mathcal{C}_2$. ■ 4.7

Clearly, taking $\mathcal{C}_1$ as the empty set and $\mathcal{C}_2$ as the set of all configurations of β_m, we recover $Sat(\phi)$.

Before obtaining our set of equations, we need some properties of the relation $\leq$ defined above.

Lemma 4.8

Let C be a configuration and $\mathcal{C}_1$, $\mathcal{C}_2$ two sets of configurations.

(1) *$C \models \Diamond\phi$ iff $C \leq Sat(\phi)$.*

(2) *$C \leq \mathcal{C}_1$ and $C \leq \mathcal{C}_2$ iff $C \leq \mathcal{C}_1 \bigtriangledown \mathcal{C}_2$, where*

$$\mathcal{C}_1 \bigtriangledown \mathcal{C}_2 = \{C_1 \cap C_2 \mid C_1 \in \mathcal{C}_1, C_2 \in \mathcal{C}_2\}.$$

(3) *$C \not\leq \mathcal{C}_1$ and $C \not\leq \mathcal{C}_2$ iff $C \not\leq \mathcal{C}_1 \cup \mathcal{C}_2$.* ■ 4.8

We have now:

Theorem 4.9 *Compositional equations for $Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$*

Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations of β_m, and let ϕ, ψ be two formulas.

$$\begin{aligned} Sat(\,\mathcal{C}_1,\, \phi \wedge \Diamond\psi,\, \mathcal{C}_2\,) &= Sat(\,\mathcal{C}_1,\, \phi,\, \mathcal{C}_2 \bigtriangledown Sat(\psi)\,) \\ Sat(\,\mathcal{C}_1,\, \phi \wedge \neg\Diamond\psi,\, \mathcal{C}_2\,) &= Sat(\,\mathcal{C}_1 \cup Sat(\psi),\, \phi,\, \mathcal{C}_2\,) \end{aligned}$$

Proof: We only prove the $\supseteq$ inclusion of the first equation. The rest is similar. Let $C \in Sat(\mathcal{C}_1, \phi, \mathcal{C}_2 \bigtriangledown Sat(\psi))$. By definition 4.7, $C \models \phi$, $C \not\leq \mathcal{C}_1$ and $C \leq \mathcal{C}_2 \bigtriangledown Sat(\psi)$. By Lemma 4.8(2), $C \leq \mathcal{C}_2$ and $C \leq Sat(\psi)$. By Lemma 4.8(1), $C \models \Diamond\psi$. So $C \in Sat(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2)$. ■ 4.9

By exhaustively applying these equations, we can express $Sat(\phi)$ in terms of sets $Sat(\gamma)$ for conjunctions of literals γ.

It is easy to adapt these equations to $Sat_f(\phi)$. Instead of doing that, we shall go one step further. Since the set $Sat_f(\phi)$ can be large, and we are only interested in deciding if it equals the empty set, it suffices to compute its largest elements.

Definition 4.10 *Last sets of configurations*

Let $max\{\mathcal{C}\}$ denote the set of maximal elements of a set of configurations $\mathcal{C}$ with respect to set inclusion.
We define $Last(\phi) = max\{Sat_f(\phi)\}$.
More generally, let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations of β_f.
We define $Last(\mathcal{C}_1, \phi, \mathcal{C}_2) = max\{Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2)\}$. ■ 4.10

Clearly, $Sat_f(\phi) = \emptyset$ if and only if $Last(\phi) = \emptyset$. Moreover, we have $Last(\phi) = Last(\emptyset, \phi, \mathcal{F})$. We obtain the following equations for $Last(\phi)$.

Theorem 4.11 *Compositional equations for $Last(\mathcal{C}_1, \phi, \mathcal{C}_2)$*

Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations of β_f, and let ϕ, ψ be two formulas. Let $\mathcal{C} = \mu S.Last(\psi) \bigtriangledown \{E_f\}$.

$$\begin{aligned} Last(\,\mathcal{C}_1,\, \phi \wedge \Diamond\psi,\, \mathcal{C}_2\,) &= Last(\,\mathcal{C}_1,\, \phi,\, max\{\mathcal{C}_2 \bigtriangledown \mathcal{C}\}\,) \\ Last(\,\mathcal{C}_1,\, \phi \wedge \neg\Diamond\psi,\, \mathcal{C}_2\,) &= Last(\, max\{\mathcal{C}_1 \cup \mathcal{C}\},\, \phi,\, \mathcal{C}_2\,) \end{aligned}$$

■ 4.11

This is the result we have been aiming for. Using these recursive equations, we can reduce the problem of deciding the emptyness of $Last(\phi)$ to the following two problems:

- Computing $\mu S.\mathcal{C} \bigtriangledown \{E_f\}$ for an arbitrary set $\mathcal{C} \subseteq \mathcal{F}$.
- Computing $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$ for $\mathcal{C}_1$, $\mathcal{C}_2 \subseteq \mathcal{F}$ and a conjunction of literals γ.

Notice that, by definition of $\bigtriangledown$, the configurations of $\mu\mathcal{S}.\mathcal{C} \bigtriangledown \{E_f\}$ are subsets of E_f, and therefore contained in the finite prefix.

We give algorithms for these two problems in the following two sections. Let us first see how to compute, assuming these algorithms are available, if the formula $\Box\Diamond s_2$ holds in the system of Figure 1.

The formula is equivalent to $\neg\Diamond\neg\Diamond s_2$. We check if $\Diamond\neg\Diamond s_2$ holds by deciding the emptyness of $Last(\neg\Diamond s_2)$.

First, we write $\neg\Diamond s_2$ as a disjunction of formulas in normal form. In this case, $\neg\Diamond s_2 \equiv (\mathbf{true} \wedge \neg\Diamond s_2)$, which is in normal form.

We compute $Last(\mathbf{true} \wedge \neg\Diamond s_2)$ by means of the second equation of Theorem 4.11. The first step is the computation of $Last(s_2)$.

We get

$$Last(s_2) = \{\ \{e_1, e_2\}, \{e_2, e_3, e_4, e_5\}\ \}$$

Then, we have to compute $\mu\mathcal{S}_f.Last(s_2) \bigtriangledown \{E_f\}$. In this case, there exists one single cut-off event (e_6) and we obtain

$$\mu\mathcal{S}_f.Last(s_2) \bigtriangledown \{E_f\} = \{\ \{e_1, e_2\}, \{e_2, e_3, e_4, e_5\}, \{e_2, e_3, e_4, e_7\}\ \} = max\{\mathcal{F}\}$$

Finally, we compute $Last(max\{\mathcal{F}\}, \mathbf{true}, \mathcal{F})$. Since $max\{\mathcal{F}\}$ is the set of the largest configurations of β_f, no configuration C of β_f satisfies $C \not\leq max\{\mathcal{F}\}$, and therefore we obtain $\emptyset$ as result. So $Last(\mathbf{true} \wedge \neg\Diamond s_2) = \emptyset$. Then, we have $\Sigma \not\models \Diamond\neg\Diamond s_2$ and, finally, $\Sigma \models \neg\Diamond\neg\Diamond s_2$.

5 Computing $\mu\mathcal{S}.\mathcal{C} \bigtriangledown \{E_f\}$

We start by showing how to compute the following mappings.

Definition 5.1 *Finite versions of the mappings $\mathcal{S}_e$ and $\mathcal{S}$*

Let $C \in \mathcal{F}$, and let e be a cut-off event. We define $\mathcal{S}_{f\,e}(C) = \mathcal{S}_e(C) \cap E_f$. Also, we define for a set of configurations $\mathcal{C}$

$$\mathcal{S}_f(\mathcal{C}) = \mathcal{C} \cup \bigcup_{e \in \mathit{Off}} \mathcal{S}_{f\,e}(\mathcal{C}).$$

where Off denotes the set of minimal cut-off events of β_m.

Finally, we define

$$\mu.\mathcal{S}_f(\mathcal{C}) = \bigcup_{n \geq 0} \mathcal{S}_f^n(\mathcal{C})$$

■ 5.1

Let I_e be an isomorphism between $\Uparrow e^0$ and $\Uparrow e$. When constructing the finite prefix β_f, it is easy to compute 'on the fly', for every cut-off event e, the pairs $(x, I_e(x))$ such that $I_e(x) \in E_f$. We start with the pairs (b, b') such that $b \in Cut([e^0])$, $b' \in Cut([e])$, $p_m(b) = p_m(b')$. Then, whenever we add a new node

$y' \in I_e(x)^\bullet$ to the prefix, we look for the node $y \in x^\bullet$ such that $p_m(y) = p_m(y')$, which exists and is unique. Then we add (y, y') to the set of pairs.

Using these pairs, $\mathcal{S}_{fe}(C)$ can be easily computed using the definition of $\mathcal{S}_e(C)$. We show now how to compute $\mu\mathcal{S}.\mathcal{C} \bigtriangledown \{E_f\}$.

Theorem 5.2

For every set $\mathcal{C} \subseteq \mathcal{F}$ and every $n \geq 0$, $\mathcal{S}^n(\mathcal{C}) \bigtriangledown \{E_f\} = \mathcal{S}_f^n(\mathcal{C})$.
In particular, $\mu\mathcal{S}.\mathcal{C} \bigtriangledown \{E_f\} = \mu\mathcal{S}_f.\mathcal{C}$. ■ 5.2

The set $\mathcal{S}_f^n(\mathcal{C})$ can be stepwisely computed for increasing values of n.
Let C be a configuration of $\mathcal{S}_f^n(\mathcal{C}) \setminus \mathcal{S}_f^{n-1}(\mathcal{C})$ for $n > 0$. Then, C is obtained by shifting some configuration $C' \subseteq E_f$ and intersecting the result with E_f. This implies $|C| \geq |C'|$.
Since E_f is finite, we eventually reach an n such that $\mathcal{S}_f^{n+1}(\mathcal{C}) = \mathcal{S}_f^n(\mathcal{C})$. Once this point is reached, the computation can terminate, because $\mathcal{S}_f^m(\mathcal{C}) = \mathcal{S}_f^n(\mathcal{C})$ for every $m \geq n$, and therefore $\mu\mathcal{S}_f.\mathcal{C} = \mathcal{S}_f^n(\mathcal{C})$.

The size of the set $\mu\mathcal{S}_f.\mathcal{C}$ can grow quickly with the number of cut-off events; in turn, this number can be high if the system is very nondeterministic. This limits the applicability of our method to systems with an small amount of nondeterminism, as is tipically the case in asynchronous circuits.

6 Computing $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$

By the definition of $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$, we have

$$Last(\mathcal{C}_1, \gamma, \mathcal{C}_2) = max\{ \bigcup_{C_2 \in \mathcal{C}_2} Last(\mathcal{C}_1, \gamma, \{C_2\}) \}.$$

Moreover, we have

$$C \in Last(\mathcal{C}_1, \gamma, \{C_2\}) \text{ iff } C \in Last(\emptyset, \gamma, \{C_2\}) \text{ and } C \not\leq C_1.$$

$C \not\leq C_1$ can be checked using the definition. So it suffices to solve the problem of computing $Last(\emptyset, \gamma, \{C_2\})$.

It is shown in [4] that the set $Last(\emptyset, \gamma, \{C_2\})$ contains at most one configuration. We show how to compute it (respectively, how to show that it does not exist) using Linear Programming. A Linear Programming problem is a set of linear inequations, or constraints, over a set of real variables, together with a linear function on the same variables called the optimization function. A feasible solution of the problem is an assignation of values to the variables which satisfies all the constraints. A feasible solution is optimal if the value of the optimization function applied to it is greater or equal than the value for any other feasible solution.

Definition 6.1 *The Linear Programming problem $L(\gamma, C)$*

Let C be a configuration of β_f and γ a conjunction of literals.
We associate to each event $e \in C$ a real variable $X(e)$. X denotes a the vector whose components are these variables.
For every condition b of $C^\bullet$, ${}^\circ b$ denotes the unique input event of b.
Similarly, for every condition b of ${}^\bullet C$, b° denotes the unique output event of b contained in C.
For every condition b of $Min(\beta_f) \cup C^\bullet$, $M(b)$ is a shortening for:

$$M(b) = \begin{cases} 1 & \text{if } b \in Min(\beta_f) \setminus {}^\bullet C \\ 1 - X(b^\circ) & \text{if } b \in Min(\beta_f) \cap {}^\bullet C \\ X({}^\circ b) & \text{if } b \in C^\bullet \setminus {}^\bullet C \\ X({}^\circ b) - X(b^\circ) & \text{if } b \in {}^\bullet C \cap C^\bullet \end{cases}$$

The Linear Programming problem $L(\gamma, C)$ consists of the following inequations:

(1) For every $e \in C$: $0 \leq X(e) \leq 1$.

(2) For every condition b of ${}^\bullet C \cap C^\bullet$: $X({}^\circ b) \geq X(b^\circ)$ (equivalently, $M(b) \geq 0$)

(3) For every literal s of γ, $\sum_{b \in B(s)} M(b) = 1$, where $B(s)$ is the set of conditions of $Min(\beta_f) \cup C^\bullet$ labelled by s.

(4) For every literal $\neg s$ of γ, $\sum_{b \in B(s)} M(b) = 0$.

and the optimization function: $\sum_{e \in C} X(e)$ ■ 6.1

By the inequations (1), the value of the optimizing function is not greater than $|C|$. It follows that if the problem has a feasible solution, then it has an optimal one.

Intuitively, the Linear Programming problem encodes in linear inequations the conditions that a vector has to satisfy in order to be the characteristic vector of a configuration satisfying γ. Also, it can be proved that the optimal solution of the problem, if it exists, is integer. Then, the group (1) ensures that the solution is in fact a characteristic vector; the group (2) that the set corresponding to the vector is left-closed, and therefore a configuration; the group (3) that the cut associated to the configuration contains some condition labelled s for every s of γ; finally, (4) ensures that this cut contains no condition labelled s for every $\neg s$ of γ.

Let us construct the system $L(s_4 \wedge \neg s_2, \{e_2, e_3, e_4, e_5\})$ for the finite prefix of Figure 3.

Group (1)	$0 \leq X(e_i) \leq 1 \quad \text{for } i = 2,3,4,5$
Group (2)	$X(e_2) \geq X(e_4)\ ,\ \ X(e_3) \geq X(e_4)\ ,\ \ X(e_4) \geq X(e_5)$
Group (3)	$1 - X(e_3) + X(e_4) - X(e_5) = 1$
Group (4)	$X(e_1) + X(e_5) = 0$
Maximize	$X(e_2) + X(e_3) + X(e_4) + X(e_5)$

The optimal solution of this problem is

$$X(e_2) = 1 \quad X(e_3) = 1 \quad X(e_4) = 1 \quad X(e_5) = 0.$$

The reader can check that $\{e_2, e_3, e_4\}$ is the largest configuration contained in $\{e_2, e_3, e_4, e_5\}$ that satisfies $s_4 \wedge \neg s_2$.

Theorem 6.2

(1) *If $L(\gamma, C_2)$ has no solution, then $Last(\emptyset, \phi, \{C_2\}) = \emptyset$.*

(2) *If $L(\gamma, C_2)$ has an optimal solution, then it is unique and it equals the characteristic vector in C_2 of the unique element of $Last(\emptyset, \phi, \{C_2\})$.*

■ 6.2

By the polynomiality of Linear Programming, $Last(\emptyset, \gamma, \{C_2\})$ can be computed in polynomial time in the size of C_2. It is well know that the simplex algorithm has better average performance than the known polynomial algorithms, in spite of having exponential worst-case complexity. Some experiments performed by Thomas Thielke[2] using simplex indicate that the computation time is approximately $O(|C_2|^{2.7})$.

7 The Deterministic Case

We summarise in this section the results of [4] for the systems in which the finite prefix β_f has exactly one maximal configuration; they are systems in which, if two transitions are simultaneously enabled, then they are concurrent (there are no conflicts). Although this is a small subclass of Petri nets, they play an important rôle in the verification of asynchronous circuits. As pointed out in [11], the transition systems of the nets of this class are semimodular Muller-diagrams, the classical formal tool for the description of self-timed circuits. This makes the class a suitable modelling tool for these circuits.
It is shown in [4] that, in this particular case, $Last(\phi)$ has at most one element for every formula ϕ in normal form, and it can be computed solving a number of Linear Programming problems linear in the length of ϕ. This result proves that our model checker has linear complexity in the length of the formula and polynomial complexity in the size of the finite prefix.

[2]Personal communication.

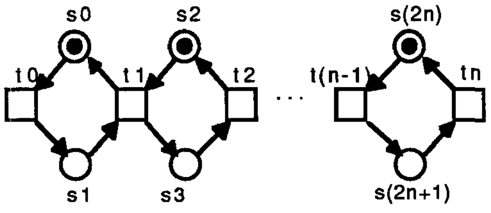

Figure 4: A simplified model of a concurrent buffer.

Conflict-free Petri nets are a class of net systems in which for every place s, $|s^\bullet| \leq 1$ or $s^\bullet \subseteq {}^\bullet s$. They have been thoroughly studied in several papers (see, for instance [6,13]).
The family of net systems shown in Figure 4 is conflict-free. They are very simple models of a concurrent buffer of length n. A token in s_{2i} means that the cell $i-1$ is empty, while a token in s_{2i+1} means it is full. Items enter the buffer through the occurrence of t_0 and leave it through the occurrence of t_n.

Using a result of [13], it is easy to prove that the finite prefix of a 1-safe conflict-free system $\Sigma = (S, T, F, M_0)$ can be constructed in $O(|T|^2 \cdot |S|^2)$. We get as corollary that our model checker has polynomial complexity in the size of the system for this class of nets, in spite of the fact that 1-safe conflict free systems may have exponential state spaces (the family of Figure 4 has). In particular, for a buffer of length n we can check in polynomial time in n whether it is possible to reach a certain state (this is what has to be done in order to check that all cells can be simultaneously full) . However, Valmari's reduced state spaces [12] cannot be used to solve these problems in polynomial time; the reason is that for every state there is a formula which is true only of that state; therefore, no reduced state space is equivalent to the full state space for this logic. The algorithm of [5] faces a similar problem: in the worst case it has to completely generate the state space before the property can be decided, and it can be exponential. Finally, the approach of [10] is not applicable.

8 Conclusions

We have shown that it is possible to design model checkers that work on a net unfolding, a well-known partial order semantics of concurrent systems very close to event structures. A model checker of this kind has also been described in [10]; however, it uses a non-standard semantics and does not handle the whole set of properties of a logic. Our verification algorithm can check several important safety properties; reachability of a marking, coverability problems and liveness of transitions. We have shown that it is polynomial in the size of the system for a non-trivial class of systems with exponential state spaces, for which the algorithms of [5,10,12] are exponential or not applicable.

Verification algorithms for interleaving semantics usually traverse the state s-pace. In our approach there exists no explicit representation of the state space; we have used a new technique, in which we only compute some maximal states (in fact, maximal configurations) using Linear Programming.
It is pointed out in [8] that coverability problems can be solved 'on the fly' – i.e. while constructing the prefix. However, there existed so far no technique to reuse the prefix, once constructed, to solve new problems. This was annoying, because the size of the prefix can be much smaller than the computation time required to construct it. Our results solve this problem.

Acknowledgement. This paper generalises previous results obtained together with Eike Best. I am also very indebted to him for many fruitful discussions.

References

[1] E. Best and J. Esparza: Model Checking of Persistent Petri Nets. Computer Science Logic 91, LNCS 626, 35–52 (1991).

[2] E. Best and C. Fernández: Nonsequential Processes – A Petri Net View. EATCS Monographs on Theoretical Computer Science Vol. 13 (1988).

[3] J. Engelfriet: Branching processes of Petri nets. Acta Informatica Vol. 28, 575–591 (1991).

[4] J. Esparza: Model Checking Using Net Unfoldings. Hildesheimer Informatikfachbericht 14/92 (October 1992).

[5] P. Godefroid and P. Wolper: Using Partial Orders for the Eficient Verification of Deadlock Freedom and Safety Properties. Computer Aided Verification, LNCS 575, 332–343 (1991).

[6] R. Howell and L. Rosier: On questions of fairness and temporal logic for conflict-free Petri nets. Advances in Petri Nets 1988, LNCS 340, 200–220, Springer, Berlin (1988).

[7] G.E. Hughes and M.J. Creswell: An Introduction to Modal Logic. Methuen and Co. (1968).

[8] K.L. McMillan: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. 4th Workshop on Computer Aided Verification, Montreal, 164–174 (1992).

[9] M. Nielsen, G. Plotkin and G. Winskel: Petri Nets, Event Structures and Domains. Theor. Comp. Sci. Vol. 13, 1, pp. 85–108 (1980).

[10] D. K. Probst and H.F. Li: Partial-Order Model Checking: A Guide for the Perplexed. Computer Aided Verification, LNCS 575, 322–332 (1991).

[11] M. Tiusanen: Some Unsolved Problems in Modelling Self-Timed Circuits Using Petri Nets. EATCS Bulletin Vol. 36, 152–160 (1988).

[12] A. Valmari: Stubborn Sets for Reduced State Space Generation. Advances in Petri Nets 1990, LNCS 483, 491–515 (1990).

[13] H. Yen: A polynomial time algorithm to decide pairwise concurrency of transitions for 1-bounded conflict-free Petri nets. Inf. Proc. Lett. 38, 71–76 (1991).