Model Checking of Persistent Petri Nets[1]

Eike Best and Javier Esparza
Institut für Informatik
Universität Hildesheim
Marienburger Platz 22
W-3200 Hildesheim, Germany
e-mail: {E.Best,esparza}@informatik.uni-hildesheim.de

Abstract

In this paper we develop a model checking algorithm which is fast in the size of the system. The class of system models we consider are safe persistent Petri nets; the logic is S_4, i.e. propositional logic with a 'some time' operator. Our algorithm does not require to construct any transition system: We reduce the model checking problem to the problem of computing certain Parikh vectors, and we show that for the class of safe marked graphs these vectors can be computed – from the structure of the Petri net – in polynomial time in the size of the system.

1 Introduction

Model checking - the algorithmic determination of truth or falsehood of a modal or temporal logic formula, given a model - faces, when applied to concurrent systems, the state explosion problem: the size of the transition system (when finite) can at best be assumed exponential in the size of the underlying system. Therefore, in order to be able to verify properties of non–toy systems, the algorithms have to be able to accept as input graphs containing millions of nodes.

Much work has been done on how to palliate this problem, following two approaches. The first is to improve the efficiency of existing general algorithms: explicit knowledge about concurrency can be used in order to obtain condensed transition systems [9,11,18]. These techniques have the advantage of being generally applicable; however, it is very difficult to know *a priori* if they will be really effective.

The second approach attemps to take advantage of special properties of the underlying model in order to speed up the model checking algorithm. This could lead to efficient, albeit special purpose methods. Two examples of this line of work are [5,15]. However, these papers also require the construction of transition systems. In this contribution, we are more radical: we investigate the possibility of obtaining model checkers which do not require at all to construct the associated transition system, i.e. model checkers that work directly on the syntax. We consider the modal logic S_4

[1]Partly supported by the Esprit Basic Research Action 3148 DEMON

[10] tailored for safe Petri nets (our syntax), and concentrate on a particular subclass of models, namely safe persistent Petri nets [13]. The logic can express properties such as reachability of a marking, liveness of a transition or mutual exclusion of a set of transitions. It also allows some "counting": properties such as "in order to reach a marking in which place s has one token, transition t has to occur 4 times" can be expressed as well.
(Safe) persistent nets are being currently used to model self-timed circuits [17]. In general, concurrent but deterministic systems (applications appear mainly in hardware design) can be modelled using persistent nets.
Given a safe persistent system Σ and a formula ϕ, we show how to reduce the model checking problem to a set of Linear Programming problems. For the subclass of safe T–systems, we prove that the model checker is polynomial in the size of Σ, although exponential in the length of ϕ. Since formulae are usually short, while systems can be very large, this is a very satisfactory result; moreover, as shown in the paper, a model checker polynomial in both the size of Σ and the length of ϕ can exist only if $P = NP$.
The paper is organised as follows. Section 2 introduces the logic. Section 3 discusses briefly the model checking problem. Section 4 introduces the models: persistent and strongly persistent systems. Section 5 presents some results on net processes; in particular, that strongly persistent systems have one single maximal process. The main theorem for the construction of the model checker is proved in Section 6. The model checker itself is described in Section 7. The particular case of T–systems is studied in Section 8. Some basic definitions are contained in an Appendix, although reading this paper is easier if the reader is familiar with the basic notions of Petri nets (otherwise, see [16]).

2 A Modal Logic for Safe Petri Nets

We define a simple modal logic over computations (more precisely occurrence sequences) of safe marked Petri nets, with the following basic propositions:

- Assertions of the form s, to be used with a model containing a place named s. The intended meaning is 'after the present computation, a token is on s'.
- Assertions of the form $t \leq 4$, to be used with a model containing a transition named t. The intended meaning is 'in the present computation t occurs no more than 4 times'.

Our logic is S_4 [10], i.e. , propositional logic augmented with the modal operator $\Diamond$, meaning 'it is possible that ... '.

Definition 2.1 *Syntax of formulae*

The formulae ϕ of our logic have the following form:

$$\begin{array}{lll} \phi ::= & \mathbf{true} & \text{(Truth)} \\ & s & \text{(Place assertion)} \\ & t \leq k \quad (k \in \mathbf{N} \cup \{-1\} \cup \{\omega\}) & \text{(Transition assertion)} \\ & \neg\phi & \text{(Negation)} \\ & \phi_1 \wedge \phi_2 & \text{(Conjunction)} \\ & \phi_1 \vee \phi_2 & \text{(Disjunction)} \\ & \Diamond\phi & \text{(Some time } \phi\text{).} \end{array}$$

A literal is a (possibly negated) place or transition assertion. A formula is called propositional iff it does not contain a modal operator. ■ 2.1

Derived formulae and operators are: $(\phi_1 \Rightarrow \phi_2) = \neg\phi_1 \vee \phi_2$; etc., and the modal operator $\Box\phi = \neg\Diamond\neg\phi$ ('always ϕ'). We also write $t > k$ for $\neg(t \leq k)$.
This logic can be interpreted on safe marked Petri nets $\Sigma = (S, T, F, M_0)$. We define what it means for Σ to satisfy a formula by defining inductively what it means that a formula is satisfied by an occurrence sequence σ. We fix some notations first: the set of all occurrence sequences from the initial marking M_0 is denoted by $\mathcal{L}(\Sigma)$, the language of Σ. For two occurrence sequences σ, τ, $\sigma \leq \tau$ if σ is a prefix of τ. Given a total ordering on the set T of transitions, the Parikh vector of an occurrence sequence σ, denoted by $\mathcal{P}(\sigma)$, is given by:

$$\mathcal{P}(\sigma)(t_i) = \text{ number of times } t_i \text{ appears in } \sigma.$$

Definition 2.2 *Satisfaction*

Let ϕ be a formula (of the above form), let $\Sigma = (S, T, F, M_0)$ be a safe marked Petri net and $\sigma \in \mathcal{L}(\Sigma)$. We define $\sigma \models \phi$ (σ satisfies ϕ) inductively:

$$\begin{array}{lll} \sigma \models \mathbf{true} & & \text{always.} \\ \sigma \models s & \text{iff} & M_0[\sigma\rangle M \wedge M(s) = 1. \\ \sigma \models t \leq k & \text{iff} & \mathcal{P}(\sigma)(t) \leq k. \\ \sigma \models \neg\phi & \text{iff} & \text{not } \sigma \models \phi. \\ \sigma \models \phi_1 \wedge \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ and } \sigma \models \phi_2. \\ \sigma \models \phi_1 \vee \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ or } \sigma \models \phi_2. \\ \sigma \models \Diamond\phi & \text{iff} & \exists \tau \geq \sigma : \tau \models \phi. \end{array}$$

Finally, $\Sigma \models \phi$ iff $\varepsilon \models \phi$ (where ε is the empty sequence). ■ 2.2

Notice that $(t \leq \omega) \Leftrightarrow \mathbf{true}$ and $(t \leq -1) \Leftrightarrow \mathbf{false}$. These formulae are introduced just out of syntactic convenience in order to write formulae in a more compact form.
The logic permits one to express safety properties such as:

Reachability of a marking. The system of figure 1 satisfies $\Diamond(\neg s_1 \wedge s_2 \wedge \neg s_3 \wedge s_4)$ iff the marking $(s_1, s_2, s_3, s_4) = (0, 1, 0, 1)$ is reachable.

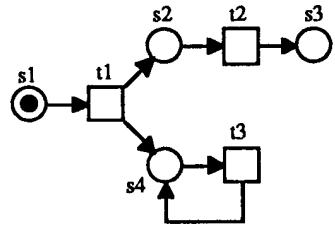


Figure 1: A safe system

- Concurrency of transitions. The system satisfies $\Diamond(s_1 \wedge s_2)$ iff transitions t_1 and t_2 are concurrently enabled at some reachable marking.
- Liveness of a transition. The system satisfies $\Box\Diamond s_4$ iff transition t_3 is live.

These properties involve place assertions only. Transition assertions permit to express properties as 'in order to reach a state with $M(s_3) = 1$, transition t_3 has to occur': $\Box(s_3 \Rightarrow t_3 > 0)$.

3 The Model Checking Problem

The Model Checking Problem (MCP) investigated in this paper is the problem of determining, given a formula ϕ in our logic and a system Σ, whether or not $\Sigma \models \phi$.

It is easy to give a lower bound on the complexity of the MCP: since the reachability problem is known to be PSPACE–complete for safe Petri nets [12], and reachability is expressible in our logic, the MCP is PSPACE–hard. Moreover, even for the simplest concurrent systems the problem is still NP–hard.

Let $i \in \mathbf{N}$ and $\Sigma_i = (S_i, T_i, F_i, M_i)$ be the system given by:

$$
\begin{aligned}
S_i &= \{s_1, \ldots, s_i\} \\
T_i &= \{t_1, \ldots, t_i\} \\
F_i &= \{(s_1, t_1), \ldots, (s_i, t_i)\} \\
M_i(s_j) &= 1 \text{ for all } j, 1 \leq j \leq i
\end{aligned}
$$

Define $\mathcal{S} = \{\Sigma_i \mid i \in \mathbf{N}\}$.

Proposition 3.1

The MCP for $\mathcal{S}$ is NP–hard.

Proof: By reduction from SAT (satisfiability of propositional logic). Let K be a propositional formula on variables $x_1, \ldots, x_k$. We construct the formula $\Diamond\phi$, where ϕ is obtained from K by replacing x_j by s_j for all $j, 1 \leq j \leq k$. It is easy to show that K is satisfiable iff $\Sigma_k \models \Diamond\phi$ (see [3]). ■ 3.1

So even for such a simple class as $\mathcal{S}$, there is little hope to find a polynomial algorithm for MCP. However, there could exist an algorithm which is exponential on the length of the formula, but polynomial in the size of the system. Such a result carries interest, because the system is usually much larger than the formula. We design in the following sections a model checking algorithm for safe (strongly) persistent systems, and show that for the subclass of safe T–systems – of which $\mathcal{S}$ is, in turn, a trivially small subclass – this type of complexity (exponentiality in the size of the formula but polynomiality in the size of the system) can be obtained.

4 Persistent and strongly persistent net systems

We are interested in the class of nets without conflicts: when two transitions are enabled at a marking, then they are concurrent. We call these systems strongly persistent. Strongly persistent systems are a subclass of the slightly larger and well known class of persistent systems [13], in which when two transitions are enabled at a marking then they can occur in any order, but not always concurrently. The model checking problem for persistent systems reduces easily to the problem for strongly persistent systems.

Definition 4.1 *Persistence and strong persistence*

(i) A Petri net Σ is persistent iff for all $M \in [M_0\rangle$ and for all $t_1, t_2 \in T, t_1 \neq t_2$, if $M[t_1\rangle$ and $M[t_2\rangle$ then $M[t_1 t_2\rangle$ and $M[t_2 t_1\rangle$.

(ii) Σ is strongly persistent iff for all $M \in [M_0\rangle$ and for all $t_1, t_2 \in T, t_1 \neq t_2$, if $M[t_1\rangle$ and $M[t_2\rangle$ then $M[\{t_1, t_2\}\rangle$. ■ 4.1

Strong persistence implies persistence, but the converse is not true.

Persistent systems can be translated into strongly persistent systems, so that a formula is true of a persistent system if and only if it is true of its translation. The proof of this result can be found in [3].

Lemma 4.2

Let Σ be a persistent system. There exist a polynomial time algorithm to construct a system Σ', with the same set of transitions as Σ, enjoying the following properties:

(i) *Σ' is strongly persistent.*

(ii) $\mathcal{L}(\Sigma) = \mathcal{L}(\Sigma')$.

(iii) *There is a function f from the formulae of Σ in the formulae of Σ' such that*
$\sigma \models F$ *in* Σ *iff* $\sigma \models f(F)$ *in* Σ'. ■ 4.2

(Strongly) persistent systems find applications in the design of switching circuits [17] and when modelling deterministic concurrent systems by means of Petri nets.

5 The lattice of cuts. Processes of strongly persistent systems

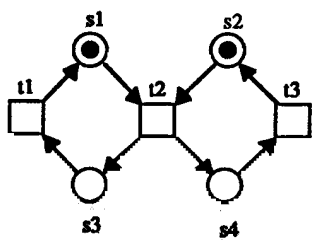
In this section, we state some elementary properties of finite occurrence nets. Basic definitions are given in the Appendix. The proofs appear either in [8] or in [3].

Throughout the section, let $N = (B, E, F)$ be a finite occurrence net, let $\mathcal{C}$ denote the set of B-cuts of N, and let $(X, \prec) = (B \cup E, F')$ be the partial order associated with N [4]. Whenever we speak of cuts in the sequel, we shall always mean B-cuts.

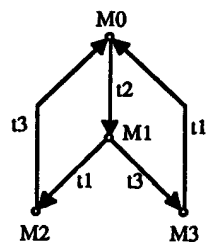
Figure 2.(a) shows a strongly persistent system; its reachability graph is shown in Figure 2.(b). Finally, Figure 2.(c) shows one of its processes and some cuts. For $c_1, c_2 \in \mathcal{C}$, let $c_1 \sqsubseteq c_2$ iff

$$\forall b_1 \in c_1 \, \forall b_2 \in c_2 : \neg(b_2 \prec b_1).$$

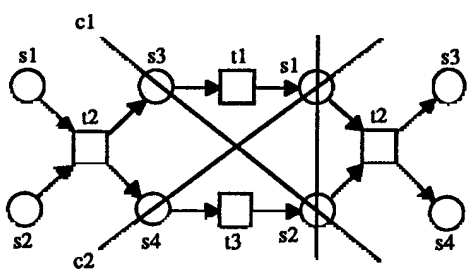
We study some properties of this definition.



(a) A strongly persistent system



(b) Its reachability graph



(c) One of its processes, and some cuts.

Figure 2: Illustration of the notion of process and cut

Lemma 5.1

$\sqsubseteq$ is transitive and antisymmetric. ■ 5.1

For $x \in B \cup E$ and $c \in \mathcal{C}$, let $x \prec c$ denote $\exists b' \in c : x \prec b'$.

Lemma 5.2

Let $c_1, c_2 \in C$. Then

(i) $c_1 \sqcup c_2 = (c_1 \cup c_2) \backslash (\{b_1 \in c_1 \mid b_1 \prec c_2\} \cup \{b_2 \in c_2 \mid b_2 \prec c_1\})$
is the lowest upper bound of c_1, c_2 with respect to $\sqsubseteq$.

(ii) $c_1 \sqcap c_2 = (c_1 \cup c_2) \backslash (\{b_1 \in c_1 \mid c_2 \prec b_1\} \cup \{b_2 \in c_2 \mid c_1 \prec b_2\})$
is the greatest lower bound of c_1, c_2 with respect to $\sqsubseteq$.

(iii) $c_1 \cap c_2 \subseteq (c_1 \sqcup c_2) \cap (c_1 \sqcap c_2)$

(iv) $c_1 \cup c_2 \supseteq (c_1 \sqcup c_2) \cup (c_1 \sqcap c_2)$

(v) $\sqcap$ *and* $\sqcup$ *are monotonic with respect to* $\sqsubseteq$. ■ 5.2

In the process of Figure 2.(c), $c = c_1 \sqcup c_2$ is the lowest upper bound of c_1 and c_2.
As a consequence of parts (i) and (ii) of this lemma, $(C, \sqcup, \sqcap)$ is a lattice. This lattice has always a least element, denoted by $Min(\pi)$, but not always a maximal one, depending on whether the process π is infinite or not. When the maximal element exists, we denote it by $Max(\pi)$. Further, $\Downarrow c$ denotes the subprocess of π below c, i.e. the elements between $Min(\pi)$ and c (inclusively). $\Uparrow c$ denotes the elements above c including c itself.
The nice property of safe strongly persistent systems we shall exploit is that they have exactly one maximal process, up to isomorphism. A process of a system Σ is called maximal if it is not isomorphic to a subprocess of a process of Σ.

Theorem 5.3 *[3]*

All maximal processes of a safe strongly persistent system are isomorphic to each other.

The uniqueness of the maximal process and the safeness of the system guarantee that, given a sequence σ, there exists a (unique) cut c of the maximal process such that $\sigma \in Lin(\Downarrow c)$. We introduce the following definition.

Definition 5.4 *The mapping Cut*

Let Σ be a safe strongly persistent system. The mapping $Cut: \mathcal{L}(\Sigma) \rightarrow C$ is defined by
$Cut(\sigma) =$ the cut c of the maximal process of Σ such that $\sigma \in Lin(\Downarrow c)$.
■ 5.4

6 Conjunctive Propositional Formulae

In this section we show that strongly persistent nets, due to Theorem 5.3, have interesting properties with respect to our logic. We start, however, with an observation valid for all safe Petri nets: occurrence sequences with the same Parikh vector satisfy the same properties.

Lemma 6.1

Let σ, τ be two occurrence sequences of a system such that $\mathcal{P}(\sigma) = \mathcal{P}(\tau)$. Then, for every formula ϕ: $\sigma \models \phi \Leftrightarrow \tau \models \phi$.

Proof: Follows easily from the definition of $\models$ and the fact that both σ and τ lead to the same marking: $M_0[\sigma\rangle M$ and $M_0[\tau\rangle M$. ■ 6.1

For the rest of the section $\Sigma = (S, T, F, M_0)$ denotes a strongly persistent safe Petri net and $\pi = (B, E, F', p)$ its unique maximal process.

Lemma 6.2

Let σ_1, σ_2 be two occurrence sequences. Let $c_1 = Cut(\sigma_1)$ and $c_2 = Cut(\sigma_2)$. If $\sigma \in Lin(\Downarrow(c_1 \sqcup c_2))$, then $\mathcal{P}(\sigma) = \max\{\mathcal{P}(\sigma_1), \mathcal{P}(\sigma_2)\}$.

Proof: (i) $\mathcal{P}(\sigma) \geq \max\{\mathcal{P}(\sigma_1), \mathcal{P}(\sigma_2)\}$.

We have:

$$\begin{aligned} \mathcal{P}(\sigma_1)(t) &= |\{e \in E \mid e \prec c_1 \wedge p(e) = t\}| \\ \mathcal{P}(\sigma_2)(t) &= |\{e \in E \mid e \prec c_2 \wedge p(e) = t\}| \\ \mathcal{P}(\sigma)(t) &= |\{e \in E \mid e \prec (c_1 \sqcup c_2) \wedge p(e) = t\}| \end{aligned}$$

It follows easily that for every transition t, $\mathcal{P}(\sigma)(t) \geq \mathcal{P}(\sigma_1)(t)$ and $\mathcal{P}(\sigma)(t) \geq \mathcal{P}(\sigma_2)(t)$. Hence, (1) holds.

(ii) $\mathcal{P}(\sigma) \leq \max\{\mathcal{P}(\sigma_1), \mathcal{P}(\sigma_2)\}$.

We prove the following claim first.

Claim. If there exist $e_1, e_2 \in E$ such that $p(e_1) = p(e_2) = t$,
then $\neg((c_2 \prec e_1 \prec c_1) \wedge (c_1 \prec e_2 \prec c_2))$.
Proof. Suppose, on the contrary, that $(c_2 \prec e_1 \prec c_1) \wedge (c_1 \prec e_2 \prec c_2)$. Then $e_1 \, co \, e_2$ and $e_1 \neq e_2$; this is because if (for instance) $e_1 \preceq e_2$, then $b_2 \prec e_1 \preceq e_2 \prec b_2'$, for some $b_2, b_2' \in c_2$, contradicting the fact that c_2 is a cut. But since $e_1 \, co \, e_2$, Theorem 3.19 of [2] shows that $t = p(e_1) = p(e_2)$ can be concurrently enabled, contradicting the safeness of Σ. *End of proof.*

The claim implies that we have either one of the following two cases:

(a) $\{e \in E \mid e \prec (c_1 \sqcup c_2) \wedge p(e) = t\} \subseteq \{e \in E \mid e \prec c_1 \wedge p(e) = t\}$
(b) $\{e \in E \mid e \prec (c_1 \sqcup c_2) \wedge p(e) = t\} \subseteq \{e \in E \mid e \prec c_2 \wedge p(e) = t\}$.

In the first case, $\mathcal{P}(\sigma) \leq \mathcal{P}(\sigma_1)$. In the second, $\mathcal{P}(\sigma) \leq \mathcal{P}(\sigma_2)$. Hence, (2) holds. ■ 6.2

As a corollary of this lemma, we obtain the following result of [13]. In fact, the result was proved there for arbitrary (strongly) persistent nets, not just safe ones.

Corollary 6.3 [13]

Let $\sigma_1, \sigma_2 \in \mathcal{L}(\Sigma)$. There exist sequences τ_1, τ_2 such that $\sigma_1\tau_1$, $\sigma_2\tau_2$ are occurrence sequences and

$$\mathcal{P}(\sigma_1\tau_1) = \mathcal{P}(\sigma_2\tau_2) = \max\{\mathcal{P}(\sigma_1), \mathcal{P}(\sigma_2)\}.$$

Proof: Let $c_1 = Cut(\sigma_1)$ and $c_2 = Cut(\sigma_2)$.
Take $\tau_1 \in Lin(\Uparrow c_1 \cap \Downarrow(c_1 \sqcup c_2))$, $\tau_2 \in Lin(\Uparrow c_2 \cap \Downarrow(c_1 \sqcup c_2))$ (the intersection is defined componentwise, and can easily be shown to be a process).
Then $\sigma_1\tau_1$, $\sigma_2\tau_2 \in Lin(\Downarrow(c_1 \sqcup c_2))$, and the result follows from Lemma 6.2.
■ 6.3

We are now ready to prove the following theorem:

Theorem 6.4

Let χ a conjunction of literals. Let $\sigma_1, \sigma_2 \in \mathcal{L}(\Sigma)$.
If $\sigma_1 \models \chi$ and $\sigma_2 \models \chi$, then for every $\sigma \in \mathcal{L}(\Sigma)$ such that $\mathcal{P}(\sigma) = \max\{\mathcal{P}(\sigma_1), \mathcal{P}(\sigma_2)\}$,

we have $\sigma \models \chi$.

Proof: By Lemma 6.1, it suffices to prove the property for a particular σ satisfying the condition on the Parikh vector. Let $c_1 = Cut(\sigma_1)$, $c_2 = Cut(\sigma_2)$. Using Lemma 6.2, we choose σ as one of the linearisations of $\Downarrow(c_1 \sqcup c_2)$. We prove the claim separately for the possible literals in χ.

(i) $(\sigma_1 \models t \leq k \wedge \sigma_2 \models t \leq k) \Rightarrow \sigma \models t \leq k$
We have $\mathcal{P}(\sigma_1)(t) \leq k$ and $\mathcal{P}(\sigma_2)(t) \leq k$.
Then $\mathcal{P}(\sigma)(t) = \max\{\mathcal{P}(\sigma_1)(t), \mathcal{P}(\sigma_2)(t)\} \leq k$.

(ii) $(\sigma_1 \models t > k \wedge \sigma_2 \models t > k) \Rightarrow \sigma \models t > k$
We have $\mathcal{P}(\sigma_1)(t) > k$ and $\mathcal{P}(\sigma_2)(t) > k$.
Then $\mathcal{P}(\sigma)(t) = \max\{\mathcal{P}(\sigma_1)(t), \mathcal{P}(\sigma_2)(t)\} > k$.

(iii) $(\sigma_1 \models s \wedge \sigma_2 \models s) \Rightarrow \sigma \models s$
Let $M_0[\sigma_1\rangle M_1$ and $M_0[\sigma_2\rangle M_2$.
Because $\sigma_1 \models s$ and $\sigma_2 \models s$, we have:

$$\exists b_1 \in c_1 : p(b_1) = s \quad \text{and} \quad \exists b_2 \in c_2 : p(b_2) = s$$

Case 1: $b_1 = b_2$.
Then $b_1 = b_2 \in c_1 \cap c_2$, and by Lemma 5.2(iii), also $b_1 = b_2 \in c_1 \sqcup c_2$.
Case 2: $b_1 \neq b_2$.
Then by the safeness of Σ (and Theorems 3.15, 3.17 and 3.19 of [2]), it cannot be the case that b_1 *co* b_2. Therefore, either $b_1 \prec b_2$ or $b_2 \prec b_1$.
In the former case, $b_2 \in c_1 \sqcup c_2$, in the latter case, $b_1 \in c_1 \sqcup c_2$.
In all cases, $\exists b \in c_1 \sqcup c_2 : p(b) = s$.

(iv) $(\sigma_1 \models \neg s \wedge \sigma_2 \models \neg s) \Rightarrow \sigma \models \neg s$
Let $M_0[\sigma_1\rangle M_1$ and $M_0[\sigma_2\rangle M_2$.
Since $\sigma_1 \models \neg s$ and $\sigma_2 \models \neg s$, we have $s \notin p(c_1)$ and $s \notin p(c_2)$. Because of $c_1 \sqcup c_2 \subseteq c_1 \cup c_2$ (Lemma 5.2(iv)), we also have $s \notin p(c_1 \sqcup c_2)$, and hence $\sigma \models \neg s$.

From (i)–(iv), it follows that if both σ_1 and σ_2 satisfy a conjunction of literals χ, then σ satisfies it as well. ■ 6.4

Remark 6.5

Theorem 6.4 is false if χ is allowed to be a disjunction. Consider the formula $\phi = s_3 \vee s_4$ in the example of Figure 2. We have $t_2t_1 \models \phi$ and $t_2t_3 \models \phi$, but $t_2t_1t_3 \not\models \phi$. ■ 6.5

We associate to a conjunction of literals χ a Parikh vector in the following way:

Definition 6.6 *The mapping $Last_\chi$*

Let χ be a conjunction of literals. The mapping $Last_\chi: T \to \mathbf{N} \cup \{-1\} \cup \{\omega\}$ (with $\omega > k$ for every $k \in \mathbf{N}$) is defined as follows:

$$Last_\chi(t) = \begin{cases} -1 & \text{if no occurrence sequence satisfies } \chi \\ \sup\{\mathcal{P}(\sigma)(t) \mid \sigma \models \chi\} & \text{otherwise} \end{cases}$$

■ 6.6

Remark 6.7

By Theorem 6.4, $Last_\chi = \sup\{\mathcal{P}(\sigma) \mid \sigma \models \chi\}$. ■ 6.7

The interest of this definition lies in the following result. Loosely speaking, $Last_\chi(t)$ indicates the maximum number of times (arbitrarily many if $Last_\chi(t) = \omega$) that transition t can occur without losing the possibility of extending the current occurrence sequence to one satisfying χ.

Lemma 6.8

Let $\sigma \in \mathcal{L}(\Sigma)$. σ can be extended to an occurrence sequence $\tau \geq \sigma$ with $\tau \models \chi$ iff $\mathcal{P}(\sigma) \leq Last_\chi$.

Proof: ($\Rightarrow$): Follows easily from the definition of $Last_\chi$.

($\Leftarrow$): If $\mathcal{P}(\sigma) = Last_\chi$, then take $\tau = \sigma$. If $\mathcal{P}(\sigma) \neq Last_\chi$ then we have:

$$
\begin{array}{l}
\mathcal{P}(\sigma) \neq Last_\chi \\
\Rightarrow \{ \text{ Remark 6.7 } \} \\
\exists \sigma' \colon \sigma' \models \chi \wedge \mathcal{P}(\sigma) \leq \mathcal{P}(\sigma') \leq Last_\chi \\
\Rightarrow \\
\max\{\mathcal{P}(\sigma), \mathcal{P}(\sigma')\} = \mathcal{P}(\sigma') \\
\Rightarrow \{ \text{ Corollary 6.3 } \} \\
\exists \sigma'' \colon \ \sigma\sigma'' \text{ is an occurrence sequence } \ \wedge \ \mathcal{P}(\sigma\sigma'') = \mathcal{P}(\sigma') \\
\Rightarrow \{\sigma' \models \chi, \text{ Lemma 6.1 } \} \\
\exists \sigma'' \colon \ \sigma\sigma'' \text{ is an occurrence sequence } \ \wedge \ \sigma'' \models \chi.
\end{array}
$$

Taking $\tau = \sigma\sigma''$, the result follows. ■ 6.8

The following theorem is the kernel of our model checker: it shows how to replace a formula with one modality by a propositional formula.

Theorem 6.9

Let Σ be a safe persistent system. For every conjunction χ of literals and every occurrence sequence σ:

$$\sigma \models \Diamond\chi \ \Leftrightarrow \ \sigma \models \bigwedge_{t \in T} t \leq Last_\chi(t).$$

Proof: By definition of $\models$, $\sigma \models \Diamond\chi$ iff there exists $\tau \geq \sigma$ such that $\tau \models \chi$. By Lemma 6.8, this is the case iff $\mathcal{P}(\sigma) \leq Last_\chi$. By definition of $\models$ again, $\mathcal{P}(\sigma) \leq Last_\chi$ iff $\sigma \models \bigwedge_{t \in T} t \leq Last_\chi(t)$. ■ 6.9

7 The Model Checker

Before presenting formally the model checker, we need to introduce a standard form for the formulae of our logic.

Definition 7.1 *Standard form*

A formula $\Diamond\phi$ is a *first-degree formula* iff ϕ contains no modalities. A formula ϕ is in *standard form* iff:

- ϕ contains no derived operators, and
- for every first-degree subformula $\Diamond\phi'$ of ϕ, ϕ' is a conjunction of literals.

■ 7.1

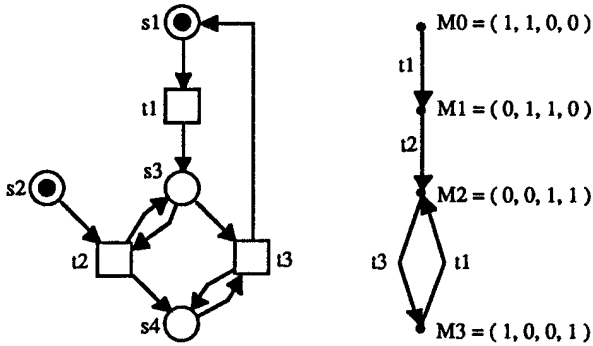


Figure 3: A system on which to use the model checker

Proposition 7.2

Every formula is equivalent to a (not necessarily unique) formula in standard form.

Proof: Easy, using propositional calculus and the schema $\Diamond(\phi_1 \vee \phi_2) = \Diamond\phi_1 \vee \Diamond\phi_2$, valid in our logic. ■ 7.2

Algorithm 7.3 *The model checker.*

Input: A safe strongly persistent system $\Sigma = (N, M_0)$ and a formula ϕ.

Output: $\Sigma \models \phi$ or $\Sigma \not\models \phi$.

```
begin
    while φ contains modalities do
        φ := φ in standard form;
        for every first-degree subformula ◇χ of φ do
            compute Last_χ;
            substitute ◇χ by ∧_{t∈T} t ≤ Last_χ(t)
        endfor
    endwhile
    (* Now φ contains no modalities *)
    check if Σ ⊨ φ using the definition and answer accordingly
end
```

■ 7.3

We apply the model checker to a small example. Consider the safe strongly persistent system Σ on the left of Figure 3.

We use the model checker to answer:

$$\Sigma \stackrel{?}{\models} \Box\,\Diamond(s_3 \wedge (s_2 \vee s_4)) \qquad (1)$$

We put the formula in (1) in standard form:

$$\neg\Diamond(\neg\Diamond(s_3 \wedge s_2) \wedge \neg\Diamond(s_3 \wedge s_4)) \tag{2}$$

Both $\Diamond(s_3 \wedge s_2)$ and $\Diamond(s_3 \wedge s_4)$ are first–degree subformulae. It is easy to see that $Last_{(s_3 \wedge s_2)} = (1,0,0)$ and $Last_{(s_3 \wedge s_4)} = (\omega,1,\omega)$. We substitute both formulae by the corresponding conjunctions of transition assertions:

$$\neg\Diamond\neg((t_1 \leq 1 \wedge t_2 \leq 0 \wedge t_3 \leq 0) \vee (t_1 \leq \omega \wedge t_2 \leq 1 \wedge t_3 \leq \omega)) \tag{3}$$

Replacing $t_1 \leq \omega$ and $t_3 \leq \omega$ by **true**, simplifying and putting the result in standard form again, we get:

$$\neg(\Diamond(t_1 > 1 \wedge t_2 > 1) \;\vee\; \Diamond(t_2 > 0 \wedge t_2 > 1) \;\vee\; \Diamond(t_3 > 0 \wedge t_2 > 1)) \tag{4}$$

There is no sequence satisfying $t_1 > 1 \wedge t_2 > 1$. Hence, the corresponding *Last* vector is $(-1,-1,-1)$, and similarly for the other two disjuncts of (4) :

$$\neg((t_1 \leq -1 \wedge t_2 \leq -1 \wedge t_3 \leq -1) \vee \ldots \vee (t_1 \leq -1 \wedge t_2 \leq -1 \wedge t_3 \leq -1)) \tag{5}$$

Replacing $t_i \leq -1$ by **false**, we get:

$$\neg(\textbf{false} \vee \textbf{false} \vee \textbf{false}) \tag{6}$$

which evaluates to **true**. Since $\varepsilon \models \textbf{true}$, Σ satisfies the original formula.

Putting ϕ in standard form can make the size of ϕ grow exponentially (this may happen, for instance, if $\phi = \Diamond\phi'$, where ϕ' is a propositional formula in conjunctive normal form). Therefore, the number of *Last* vectors to be computed is in the worst case exponential in the length of ϕ (and independent of the size of Σ).

Our model checker is completely specified only after giving the description of a procedure for computing this vector. This is done in the next section for the class of safe T–systems (which can easily be shown to be persistent).

8 Computing Last_χ for safe T–systems

Definition 8.1

A net (S,T,F) is a *T*–net iff for every $s \in S$: $|{}^\bullet s| \leq 1$ and $|s^\bullet| \leq 1$.
(S,T,F,M_0) is a *T*–system iff (S,T,F) is a *T*–net. ■ 8.1

Throughout this section, $\Sigma = (S,T,F,M_0)$ is a safe *T*–system, and C the incidence matrix of (S,T,F), which we suppose to be weakly connected (this constraint is introduced to simplify the presentation; the results can be extended to non–connected *T*-systems by computing their connected components first, and considering them separately). We shall make use of some results on *T*–systems that are presented now. They are immediate consequences of results of [7,14].

Theorem 8.2

Let T_d be the set of transitions t of Σ that do not appear in any sequence of $\mathcal{L}(\Sigma)$. X is an integer solution of the system of linear (in)equalities

$$\begin{aligned} M_0 + C \cdot X &\geq 0 \\ \forall t \in T_d: X(t) &= 0 \\ X &\geq 0 \end{aligned}$$

iff there exists an occurrence sequence σ such that $\mathcal{P}(\sigma) = X$. ■ 8.2

The constraints on the transitions of T_d are necessary. If they are suppressed then, for instance, $X = (1)$ is a solution of the equation system corresponding to the T-net $(\{s\}, \{t\}, \{(s,t),(t,s)\})$ with marking $M_0(s) = 0$. However, there is no occurrence sequence with X as Parikh vector. Once the constraint $X(t) = 0$ is added, the only solution is $X = (0)$, which corresponds to the empty occurrence sequence.

The set T_d can be very easily computed in polynomial time in the size of Σ, as shown in [3].

The computation of $Last_\chi$ can now be reduced to the solution of a Linear Programming problem. Let us see first how to associate linear constraints to the basic propositions of our logic.

Let χ be a conjunction $\chi_1 \wedge \chi_2 \ldots \wedge \chi_n$ of literals. The system of inequalities S_χ is obtained by adding to the system of Theorem 8.2 a linear constraint for each literal χ_i in the following way:

(1) If $\chi_i = s$, then add $(M_0 + C \cdot X)(s) = 1$.

(2) If $\chi_i = \neg s$, then add $(M_0 + C \cdot X)(s) = 0$.

(3) If $\chi_i = t \leq k$ then add $X(t) \leq k$.

(4) If $\chi_i = t > k$ then add $X(t) > k$.

S_χ has the following properties:

Lemma 8.3

(i) If $\sigma \models \chi$, then $\mathcal{P}(\sigma)$ is solution of S_χ.

(ii) If X is solution of S_χ, then there exists σ such that $\mathcal{P}(\sigma) = X$ and $\sigma \models \chi$.

(iii) If S_χ has infinitely many solutions, then for every $k \in \mathbf{N}$ there exists a solution $X \geq \vec{k} = (k, k, \ldots, k)$.

Proof: (i) and (ii) follow from the definition of $\models$ and Theorem 8.2. (iii) follows from results of [7], taking into account that we only consider connected T-systems. ■ 8.3

We define the Linear Programming problem LP_χ

$$\begin{array}{ll} \text{maximise} & \sum_{t\in T} X(t) \\ \text{subject to} & S_\chi \end{array}$$

LP_χ has the following property:

Lemma 8.4

Optimal solutions of LP_χ are integer.

Proof: S_χ can be written in compact form in the following way:

$$\begin{array}{l} P_1 \le M_0 + C \cdot X \le P_2 \\ T_1 \le X \le T_2 \end{array}$$

for adequate vectors P_1, P_2, T_1, T_2. In particular, $\vec{0} \le P_1 \le P_2 \le \vec{1}$; T_2 may have ω–components, meaning that there is no upper bound for the corresponding component of X.

We show that if X is a solution of S_χ, so is $\lceil X \rceil$. The lemma then follows from

$$\sum_{t\in T} X(t) \le \sum_{t\in T} \lceil X(t) \rceil .$$

Let $M_1 = M_0 + C \cdot X$ and $M_2 = M_0 + C \cdot \lceil X \rceil$. Let s be a place. We have $|{}^\bullet s| \le 1$ and $|s^\bullet| \le 1$. Then, taking into account that P_1, P_2 are vectors over $\{0,1\}$, we have

$$P_1(s) \le \lfloor M_1(s) \rfloor \le M_2(s) \le \lceil M_1(s) \rceil \le P_2(s).$$

Moreover, since T_1, T_2 are vectors on $\mathbf{N} \cup \{\omega\}$, we have $T_1 \le X \le \lceil X \rceil \le T_2$. ■ 8.4

We can now give the following computational characterisation of $Last_\chi$.

Theorem 8.5

Let Σ be a safe T–system and χ as above.

(i) If LP_χ has no solution, then $Last_\chi = -\vec{1}$.

(ii) If LP_χ has solutions but no optimal solution, then $Last_\chi = \vec{\omega}$.

(iii) If LP_χ has an optimal solution X then $Last_\chi = X$.

Proof: (*i*) If LP_χ has no solution, then by Lemma 8.3(1) no occurrence sequence satisfies χ. By definition, $Last_\chi = -\vec{1}$.

(*ii*) In this case, S_χ has infinitely many solutions. By Lemma 8.3(3), there exists for every $k \in \mathbf{N}$ a solution σ with $\mathcal{P}(\sigma) \ge \vec{k}$. By Lemma 8.3(2), $Last_\chi > \vec{k}$ for every $k \in \mathbf{N}$. Hence, $Last_\chi = \vec{\omega}$.

(*iii*) By Lemma 8.4, X is integer. By Lemma 8.3(2), there exists σ such that $\mathcal{P}(\sigma) = X$ and $\sigma \models \chi$. By the optimality of X, there is no $\tau \ge \sigma$ such that $\tau \models \chi$. By Lemma 6.8, $Last_\chi = X$. ■ 8.5

Since Linear Programming is known to be polynomial in the size of the equation system, Theorem 8.5 implies that the computation of $Last_\chi$ is polynomial in the size of the T–system. Therefore, our model checker is polynomial in the size of the system (although it has exponential worst-case complexity in the length of the formula).
It is well known that simplex seems to have better average complexity than the existing polynomial algorithms for Linear Programming, and therefore it is the algorithm that should be used in practice.

9 Conclusions

We have tailored the modal logic S_4 for safe Petri nets. The resulting logic is rather modest; for instance, it is not possible to express liveness properties (such as 'the system will eventually reach a state with one token in place s') but many useful safety properties (reachability, reversibility, liveness of transitions) can be expressed, and the logic has some counting power. We have proved that even for this simple logic (in fact, for any logic extending the propositional calculus) and the simplest classes of concurrent systems, the model checking problem is NP–hard. This shows that model checkers polynomial in both the size of the system and the length of the formula are very unlikely to exist: however, there can still be model checkers exponential in the length of the formula, but polynomial in the size of the system.

We have designed a model checking algorithm for safe strongly persistent systems; the model checking problem is reduced to obtaining a set of so called *Last* vectors, whose number can be exponential in the length of the formula. We have shown that for the class of safe T–systems, the *Last* vectors can be computed solving a Linear Programming problem, in polynomial time in the size of the system. Therefore, for this class we have obtained a model checker with the complexity mentioned above. This is the first time, to the best of our knowledge, that such a result is obtained for a non–trivial class of systems and a non–trivial logic.

Classical results (see, for instance, [7]) showed that particular properties such as liveness or reachability could be verified in polynomial time for T–systems. These results are subsumed by our paper (for the safe case); we have shown that the polynomiality result can be extended to the whole class of properties expressible by a logic.

Our results are also related to the the constrained expression formalism of [1]. In this framework, linear constrains on the behaviour of the system are obtained from its structure. Then, Integer Linear Programming is used as a decision algorithm to verify properties. In this paper, we have shown how this method can be improved for the particular class of T–systems. In particular, Integer Linear Programming, which is known to be NP-complete, can be replaced by Linear Programming. Linear Programming is also used in the work of [6] on semidecision algorithms of some first–order assertions.

Acknowledgement

The authors wish to thank Raymond Devillers for very careful reading of the text.

References

[1] G.S. Avrunin, U.A. Buy, J.C. Corbett, L.K. Dillon and J.C. Wileden: Automated Analysis of Concurrent Systems with the Constrained Expression Toolset. COINS Technical Report 90–116, University of Massachussets at Amherst (1990).

[2] E. Best and R. Devillers: Sequential and Concurrent Behaviour in Petri Net Theory. TCS Vol. 55, 87–136 (1987).

[3] E. Best and J. Esparza: Model Checking of Persistent Petri Nets. Hildesheimer Informatik Fachbericht 11/91 (1991).

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

[5] E.M. Clarke, O. Grumberg, M.C. Browne: Reasoning about Networks with many identical finite–state processes. Proceedings of the Fifth Annual Symposium on Principles of Distributed Computing, 240–248 (1986).

[6] J.M. Colom: Structural Analysis Techniques Based on Linear Programming and Convex Geometry. Ph. D. Thesis, University of Zaragoza (1990) (in spanish).

[7] F. Commoner, A.W. Holt, S. Even and A. Pnueli: Marked Directed Graphs. Journal of Computer and System Science Vol.5, 511–523 (1971).

[8] C. Fernández, M. Nielsen and P.S. Thiagarajan: Notions of Realisable Non–Sequential Processes. Fundamenta Informaticae IX, 421–454 (1986).

[9] P. Godefroid: Using Partial Orders to Improve Automatic Verification Methods. Proc. of Computer-Aided Verification Workshop, Rutgers, New Jersey (1990).

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

[11] R. Janicki and M. Koutny: Optimal Simulation for the Verification of Concurrent Systems. Technical report, McMaster University (1989).

[12] M. Jantzen: Complexity of Place/Transition Nets. Petri Nets: Central Models and Their Properties. W. Brauer, W. Reisig, G. Rozenberg (eds.), LNCS 254, 397–412 (1987).

[13] L.H. Landweber and E.L. Robertson: Properties of Conflict–free and Persistent Petri Nets. JACM Vol.25, No.3, 352–364 (1978).

[14] T. Murata: Petri Nets: Properties, Analysis and Applications. Proc. of the IEEE Vol. 77, No. 4, 541–580 (1989).

[15] H. Qin: Efficient Verification of Determinate Processes. CONCUR'91, J.C.M. Baeten, J.F. Groote (eds.), LNCS 527, 470–479 (1991).

[16] W. Reisig: Petri Nets – an Introduction. EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer Verlag (1985).

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

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

Appendix: Basic Notions

An occurrence net $N = (B, E, F')$ is an acyclic net without branched places, i.e., $F'^{+} \cap (F'^{-1})^{+} = \emptyset$ (acyclicity) and $\forall b \in B: |{}^\bullet b| \leq 1 \wedge |b^\bullet| \leq 1$ (no branching of places). Elements of E are called events and elements of B are called conditions. An occurrence net may be infinite, or may contain isolated conditions (but no isolated events). To every occurrence net, a poset $(X, \preceq) = (B \cup E, F'^{*})$ can be associated.
We denote $li = \preceq \cup \succeq$ and $co = ((X \times X) \setminus li) \cup id|_X$. $c \subseteq X$ is called a co-set iff any two elements in c are unordered, i.e. in relation co.
A cut $c \subseteq X$ is a maximal co-set. A cut c is called B-cut iff $c \subseteq B$.

$Min(N)$ is defined as $\{x \in X \mid {}^\bullet x \cap X = \emptyset\}$; $Max(N)$ is defined similarly.

A process $\pi = (N, p) = (B, E, F', p)$ of a marked net $\Sigma = (S, T, F, M_0)$ consists of an occurrence net $N = (B, E, F')$ together with a labelling $p: B \cup E \to S \cup T$ which satisfy appropriate properties such that π can be interpreted as a concurrent run of Σ. To be a process of Σ, π must satisfy the following properties:

(i) $p(B) \subseteq S$ and $p(E) \subseteq T$.

(ii) $\forall x \in B \cup E: |{\downarrow}x| \in \mathbf{N}$ (this implies that $Min(N)$ is a cut).

(iii) $\forall e \in E: p({}^\bullet e) = {}^\bullet p(e), |p({}^\bullet e)| = |{}^\bullet p(e)|$ and $p(e^\bullet) = (p(e))^\bullet, |p(e^\bullet)| = |(p(e))^\bullet|$
(transition environments are respected).

(iv) $\forall s \in S: M_0(s) = |p^{-1}(s) \cap Min(N)|$ (i.e., $Min(N)$ corresponds to the initial marking M_0).

$Lin(\pi)$ denotes the set of occurrence sequences which are linearisations of π. For a detailed explanation of these notions (and a proof that the set $Lin(\pi)$ is always nonempty), the reader is referred to [2]. An example is given in the paper (Figure 2(c)).